A lambda-calculus that achieves full laziness with spine duplication

Sherratt, David

Award date:
2019

A lambda-calculus that achieves full laziness with spine duplication

submitted by

David Rhys Sherratt

for the degree of Doctor of Philosophy

of the

University of Bath

Department of Computer Science

March 2019

Summary

This work introduces a variant of Gundersen, Heijltjes, and Parigot's atomic lambda calculus. This typeable calculus with explicit sharing extends the Curry-Howard interpretation of open deduction already established by the mentioned. We extend the intuitionisitc proof system with a switch inference rule, which corresponds to an end-of-scope operator i.e. the switch rule can only be applied to subterms of an abstraction that do not have the binding variable occuring freely inside. Combining this notion of scope with the previous calculus, where duplication of terms is performed atomically, results in the ability to duplicate the spine of an abstraction, where the spine is the direct path from the binder to the bound variable. Spine duplication has been witnessed previously with strategies involving graphs and labels, but this work uses environments and remains in a typed setting. We prove that this calculus preserves strong normalisation with respect to the lambda calculus, satisfies the confluence property, and can duplicate the spine on any abstraction.

Acknowledgments

I would like to thank my supervisor, Willem Heijltjes (*Hell-Juice*), for taking me in and supporting me throughout this strenuous part of my life. You always gave me the push I needed to improve and I will always be grateful for your time and effort. Thank you for believing in me.

I would also like to greatly thank my examiners Delia Kesner and Guy McCusker, for the surprisingly fun but interesting discussions held at the defense of this thesis.

This work was raised from the foundations provided by Willem Heijltjes, Tom Gundersen and Michel Parigot. Your contributions to the field are invaluable.

I am deeply grateful to Alessio Guglielmi (*Big Alessio, but not big as in hefty, hench, or superior*). You have always been a reliable source of advice, guidance, and constructive criticism. Because of you, I will never cut my potatoes with a knife in Germany.

I would like to thank my amazing previous and current colleagues; Andrea Aler Tubella, Chris Barrett, Valentin Blot, Paola Bruscoli, Cillian Dudley, Giulio Guerrieri, John Power, and Ben Ralph. You supported me greatly and were always willing to help me. I would like to include a special note of thanks to my older and wiser academic sister, Fanny He, who introduced me to the λ-calculus when starting my PhD. You are an incredible influence to my research. And I would not have achieved what I have without Alessio Santamaria (*Little Alessio*), the most English foreigner I have ever had the pleasure of meeting.

I had the pleasure of working with Marco Solieri during his short time at Bath, studying the graphical variant of the atomic λ-calculus. It is a regret that only a crumb of that work has made it into this thesis. Opportunities to attend workshops and present my research were also available because of Anupam Das, thank you.

Without the continuous encouragement and kindness from Arnold Beckmann, I would not be where I am today. You gave me the courage for this adventure.

Fe hoffwn i ddiolch fy ffrindiau o'r cartref; Aled, David, David, David, Jack, Jason, Jordan, Lisa, Naomi, Sam a fy nheulu; Mam, Tad, a Robbie fy mrawd mawr. O achos ichi, dwi'n berson mwy hyderus.

Zijian, I lack the words to express my true gratitude, but your support throughout most of this journey is irreplaceable.

And finally to me
- without whom this book would not have been possible.

Contents

Chapter 1

Introduction

无理取闹
wú lǐ qǔ nào
To create problems for no reason

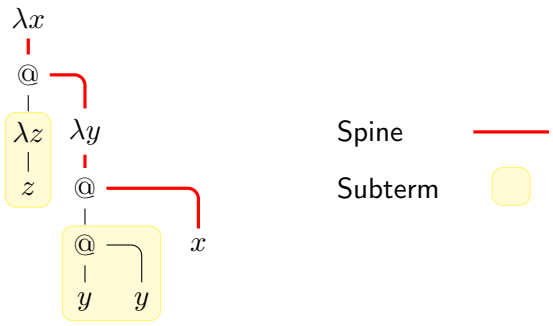
The main interest of this work is of the following *switch* rule for intuitionistic logic.

$$\frac{(A \to B) \wedge C}{A \to (B \wedge C)}\, s$$

The switch rule we are interested in has been seen in the following deep inference proof system [Gug07]. Furthermore, there do exist other switch rules for deep inference systems of intuitionistic logic that work with different connectives [Tiu06, GS14]. The switch rule has been mentioned previously not in the context of deep inference. The switch rule for multiplicative linear logic was studied in [CS97] (called "weak distributivity") and further studied in [DP04]. In weak distributivity, the switch rule can be seen for tensor ($\otimes$) and par ($\oplus$), i.e. the maps $A \otimes (B \oplus C) \to (A \otimes B) \oplus C$ and $A \otimes (B \oplus C) \to B \oplus (A \otimes C)$.

We are interested in this particular rule mentioned as it can be interpreted as an *end-of-scope* operator in the λ-calculus. Such operators have been explored in [BF82, HvO03]. This observation is fundamental of this work and explained more concretely in Chapter 2, but to give an intuition suppose we have a term $\lambda x.t$ where x is of type A and let s be a subterm of t of type C. Reading the rule upwards, the switch rule is able to split the parts of the term in the scope of x ('the B part', as it were) and subterms (such as s) that do not contain x as a free variable ('the C part'). With the use of the switch rule, we can identify the subterms of an abstraction $\lambda x.t$ that do not contain x as a free variable.

This explicit notion of scope in the λ-calculus allows us to identify the *spine* of an abstraction. The spine of an abstraction are the direct paths from the binder to bound variables. By identifying the largest subterms in $\lambda x.t$ that do not contain the variable x, we consequently identify the direct path from λx to all occurrences of x. The graph below provides an example of this for the term $\lambda x.(\lambda z.z)\, \lambda y.(y\, y)\, x$, where the spine of λx is the very thick red line and the largest subterms that would be identified by an end-of-scope operator/the switch rule are enclosed by yellow boxes.



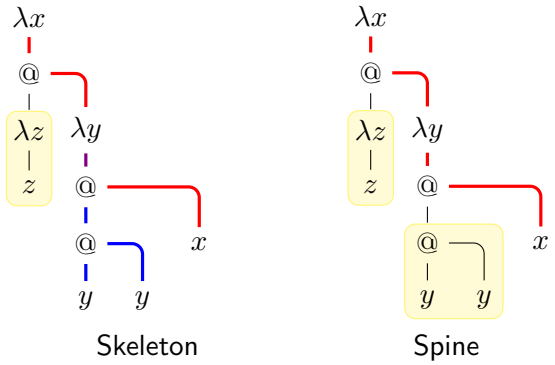
Identifying the spine of terms was implemented using graphs and labels by Blanc, Lévy, and Maranget in [BLM07], and further studied by Balabonski in [Bal12]. The latter shows that when sharing a subterm, the ability to duplicate just the spine of the subterm allows for an optimal reduction in the sense of Lévy [Lév80] for weak reduction [CH98], where a β-reduction $(\lambda x.t)\, s$ occuring in a subterm u can only reduce if the all free variables in the redex are also free in the term u. Given the restriction that u is a closed term, this is then the same as closed reduction [FM99, FMS05a].

The switch is one of two characteristic inference rules in deep inference. The other rule is the *medial*.

$$\frac{(A \vee B) \to (C \wedge D)}{(A \to C) \wedge (B \to D)}\, m$$

The computational interpretation of this medial rule was investigated in [GHP13], where the authors provide a proof system for intuitionistic logic and a Curry-Howard correspondence to a variant of the λ-calculus, *the atomic λ-calculus*. The Curry-Howard correspondence means derivations corresponds to terms and proof normalisation corresponds to term reduction. In this work we extend the proof system with this switch rule, and introduce a new variant of the calculus that corresponds with the new proof system.

The interpretation of the switch rule as a computational component and the interaction between switch and medial, results in a term calculus with explicit sharing that naturally captures this ability to duplicate the spine of a term, based on a Curry-Howard interpretation of a deep inference proof system for intuitionistic logic that uses both the presented medial and the switch rule. This new calculus differs from the result of [GHP13], which duplicates the skeleton of an abstraction (the union of all the spines of abstractions on the spine, or the *iterated spine*). It should be noted that 'skeleton' and 'spine' is terminology taken from [Bal12], and this notion of spine is discussed in [KvOvR93] called 'scope', but we use *spine* to avoid ambiguity with the natural meaning of *scope* in a term calculus. The following graph highlights the skeleton and spine of the term as before, where the skeleton of λx is composed of the very thick lines, the red lines being the spine of λx, the blue lines being the spine of λy, and purple where the spines overlap.



Skeleton Spine

Thesis outline

The main contribution of this work is a refinement of the atomic λ-calculus that has more control over duplication. Furthermore we extend the Curry-Howard correspondence between deep inference formalisms and term calculi, by looking at the computational interpretation of the switch rule. We show that our refined calculus preserves strong normalisation with respect to the λ-calculus (PSN), with the ability to duplicate the spine of a term.

In the rest of this chapter we discuss the related works for the ideas behind this dissertation. In Chapter 2 we discuss the switch rule and its Curry-Howard interpretation, an explicit end-of-scope operator, and we go over a few simple examples. In Chapter 3 we formally introduce a new variant of the atomic λ-calculus, the *spinal atomic λ-calculus* (Λ_a^S). We introduce the syntax, compilation and readback translations to the λ-calculus, and operations for the calculus (*substitution*, *book-keeping* and *exorcisms*). We also give the typing system in open deduction for the calculus. In Chapter 4 we introduce the weakening calculus, a 'middle' calculus between the atomic λ-calculus and the λ-calculus. We define the compilation and readback functions from our calculus to the weakening calculus, and describe the relationship and connections between the reduction rules in each calculus. In Chapter 5 we prove that the sharing reduction rules of our calculus are strongly normalising and confluent. We do this by invoking a measure on atomic terms, where the measure is based on the equivalent term in the weakening calculus obtained via translation. In Chapter 6 we prove that our calculus satisfies PSN with respect to the λ-calculus, i.e. for any term in the λ-calculus that is strongly normalising, the resulting term in the atomic λ-calculus obtained from translation is also strongly normalising. Lastly before our conclusion, in Chapter 7 we focus on spine duplication, and prove that our calculus is capable of duplicating the spine and only the spine of an abstraction, and leaving the maximal subterms not connected to the spine shared, achieving full laziness.

1.1 Motivation and Related Work

In this section we discuss the related work of this dissertation in detail, and describe the details of the fundamental ideas behind this work i.e. how the switch rule corresponds to an explicit end-of-scope operator and how the interaction between the switch rule and the medial rule in [GHP13] allows for spine duplication.

1.1.1 The λ-Calculus

The λ-calculus was introduced by Church in [Chu41] as an approach to formalise the foundations of mathematics. It is known for being "*the world's smallest universal programming language*" [Roj15]. Because of this, many functional programming languages are built based on the λ-calculus. A functional program consists of an expression M (representing both the algorithm and the inputs), which is then "computed" via term rewrite rules, $M \rightsquigarrow M'$ where M' is the resulting expression. These expressions are defined recursively with the syntax

$$M \quad ::= \quad x \quad | \quad \lambda x.M \quad | \quad M\,M$$

where (from left to right) we have variable, abstraction and application. Computation in the λ-calculus is implemented with $beta$-reduction. Reduction is nothing more than textual replacement of a formal parameter in the body of a function by the actual parameter supplied, and is formally shown below.

$$(\lambda x.M)\,N \rightsquigarrow_\beta M\{N/x\}$$

Types were first proposed by Russell in his development of mathematical logic [WR28], and were used as a solution to the well known Russell's paradox. The idea of types for the λ-calculus is simple. Every λ-term is a function, and its type informs us what kind of input it expects and what kind of output it will return. The grammar of types is given by

$$A \quad ::= \quad \circ \quad | \quad A \to A$$

Here, $\circ$ is a fixed type called base type. The grammar for terms in the typed λ-calculus is almost the same as before, the difference is that we label abstracted variables with their type i.e. $\lambda x^A.M$.Some terms will not have a type, and is not considered a valid term in this calculus. Once we know what type each free variable of a term has, the term will have at most one type. Types give us information about the behaviour of functions. A term is *well-typed* if all of the abstractions-applications within it are sensible i.e. if you apply a function of type $A \to B$, the argument you give it must have type A. It is also true that substitution and β-reduction preserve the typing of terms.

A consequence of using types is that self-application is no longer possible i.e. the term Ω cannot be typed. This is because the types introduce a stratification of terms: terms of type A can be supplied to terms of type $A \to B$. A term cannot have both these types at the same time, i.e. $\Omega = ww = (\lambda x.x\,x)\,\lambda x.x\,x$ cannot be typed as $w^{A\to B}w^A$ as the types of w need to be consistent. This constraint means that all typed λ-terms are *strongly normalising*. A term M is called strongly normalising if there is a bound k and a term N

which is in normal form of M such that every sequence $M \rightsquigarrow_\beta M_1 \rightsquigarrow_\beta M_2 \rightsquigarrow_\beta \cdots \rightsquigarrow_\beta N$ has at most k steps. The typed λ-calculus is closely related to proof theory and mathematical logic via the Curry-Howard correspondence [How80].

1.1.2 Deep Inference

Deep inference is a methodology for designing proof systems that can be argued to have less syntactic bureaucracy compared to more traditional formlisms [BL05, GGP10] and which allows for shorter proofs [Das14b] and to express logics that were not expressible before (for example modal logics in [SS05, Sto07]). Deep inference is a general idea in structural proof theory that is an alternative approach compared to Gentzen's formalisms i.e. the sequent calculus and natural deduction. Deep inference was first published in 2001 [GS01, BT01], and there are at least two formalisms have been designed and developed in the style of deep inference: the *calculus of structures* introduced in [Gug07] and *open deduction* [GGP10]. A third approach introduces deep inference features into a more traditional Gentzen formalism [Brü06, Brü10].

The main difference between deep inference and Gentzen's formalisms involve two principles. First we have that inference rules can be applied 'deeply'. There is no longer an idea of a main connective, nor is there anymore a distinction between meta-level and object-level. This is the main principle for the calculus of structures. Second is that proofs can be horizontally composed by the same logical connectives that are used to compose formulae. This is the main principle for open deduction. Both approaches reject the idea of tree-shape proofs of Gentzen's deductions for derivations with a single premise and conclusion. The syntax used to define these systems is similar to that of categorical logic [Hug04], but the aims and principles of proof theory are different since we study proof normalisation [AT17] and proof complexity [Das14a].

This thesis focuses on open deduction. We discuss the following example to show how we apply logical connectives at the level of derivations as well as formulas. Let there be inference rule r_1 where the premise is the formula A and the conclusion is C and an inference rule r_2 where the premise is B and the conclusion D as shown below.

$$\frac{A}{C}\,r_1 \qquad \frac{B}{D}\,r_2$$

We then apply these rules to the formula $A \wedge B$ to obtain a derivation with conclusion $C \wedge D$. In calculus of structures, we need to decide which rule ro apply first. The derivations below are not considered syntactically equal, even though they have the same premise and conclusion, because they apply the inference rules in a different order.

In open deduction, we can generalise these two derivations so that they can be considered syntactically equal. These two derivations are harmonised into one derivation, where the conjunction is applied to the two derivations. This motivates the general idea of composing proofs together.

$$\frac{\dfrac{A \wedge B}{A \wedge D}\,r_1}{C \wedge D}\,r_2 \qquad \neq \qquad \frac{\dfrac{A \wedge B}{C \wedge B}\,r_2}{C \wedge D}\,r_1$$

$$\frac{A}{C}\,r_1 \wedge \frac{B}{D}\,r_2$$

We now give a more formal definition of a derivation in open deduction, with a premise A and conclusion C. We work modulo symmetry, associativity, and unit laws of conjunction. A derivation is constructed with the following syntax.

$$\begin{array}{c} A \\ \Downarrow \\ C \end{array} \quad ::= \quad A \quad | \quad \begin{array}{c} A_1 \\ \Downarrow \\ C_1 \end{array} \wedge \begin{array}{c} A_2 \\ \Downarrow \\ C_2 \end{array} \quad | \quad \begin{array}{c} C_1 \\ \Downarrow \\ A_1 \end{array} \to \begin{array}{c} A_2 \\ \Downarrow \\ C_2 \end{array} \quad | \quad \begin{array}{c} A \\ \Downarrow \\ \dfrac{B_1}{B_2}\, r \\ \Downarrow \\ C \end{array}$$

where from left to right, (1) the premise and the conclusion can be the same formula i.e. $A = C$. (2) We can compose derivations horizontally with a conjunction $\wedge$, where $A = A_1 \wedge A_2$ and $C = C_1 \wedge C_2$. (3) We can compose derivations horizonstally with an implication $\to$ where $A = A_1 \to A_2$ and $C = C_1 \to C_2$, note that the derivation on the antecedent of the implication is inverted, it can be interpreted as a derivation where we treat the premise as the conclusion and the conclusion as the premise. Lastly (4) derivations can be composed vertically with an inference rule r from B_1 to B_2. Additionally the generic vertical composition of two derivations (without a mediating rule) exists as a derived operation in [GGP10].

A *proof system* in open deduction is given by *connectives* and *inference rules*, where we need to know how the connectives compose derivations as well as formulas (i.e. their arguments must have an "up/down" or "covariant/contravariant" direction). The inference rules we are interested in are exactly those that will help us type the λ-calculus.

1.1.3 The Atomic λ-Calculus

The atomic λ-calculus, presented in [GHP13], is an extension of the ordinary λ-calculus that include closure constructs in an explicit-substitution-like syntax. We use open deduction derivations to type the terms of the calculus. The basic inference rules obtained by embedding the usual natural deduction systems for the λ-calculus into open deduction are presented below, and are abstraction, application and (n-ary) contraction from left to right.

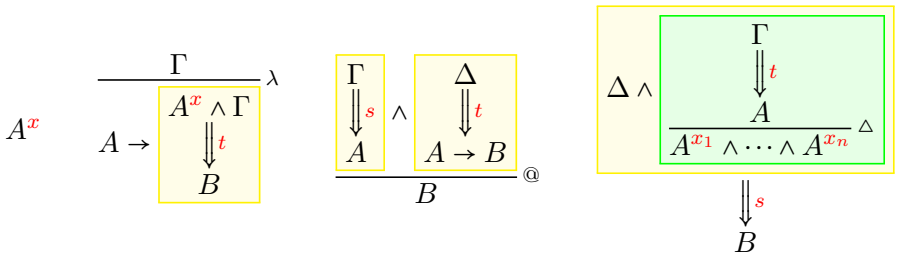
$$\frac{B}{A \to (B \wedge A)}\,\lambda \qquad \frac{A \wedge (A \to B)}{B}\,@ \qquad \frac{A}{A \wedge \cdots \wedge A}\,\triangle$$

These rules are used to type terms of the *basic calculus*, given by the grammar

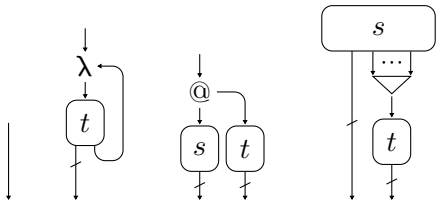
$$s, t \quad ::= \quad x \quad | \quad \lambda x.t \quad | \quad s\,t \quad | \quad s[x_1, \ldots, x_n \leftarrow t]$$

where the four constructors are called, from left to right, variable, abstraction, application and sharing. This is a *linear* calculus, so each variable occurs exactly once, and a sharing construct is used to represent multiple occurrences of a variable (or term). The variable bound by an abstraction must occur within the body of the abstraction i.e. in the term $\lambda x.t$, $x \in (t)_{fv}$. Lastly and similarly, each variable bound by the sharing construct must occur and become bound i.e. in the term $s[x_1, \ldots, x_n \leftarrow t]$, each $x_i \in (u)_{fv}$ for all $1 \leq i \leq n$. The typing derivations in open deduction for this calculus is displayed below, where the corresponding types and derivations for terms are in red. The formula A (and B) is defined

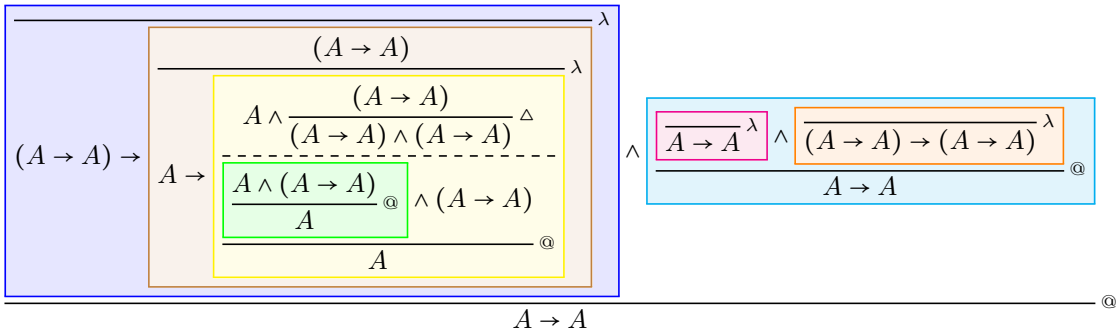
by the grammar of types discussed previously in Section 1.1.1 and Γ (and Δ) is either a conjunction of zero or more clauses where each clause is defined by A. We add the boxes to aid the reader see the horizontal composition of derivations. The colour of the box has no meaning other than to help identify derivations.

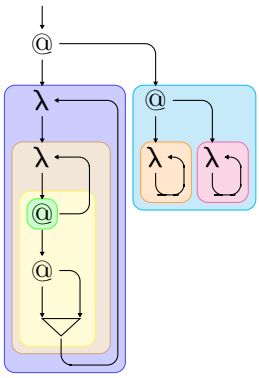


The terms can also be interpreted graphically, as displayed below. Notice that the derivations are rotated 180 degrees compared to the graphical depictions of the terms.

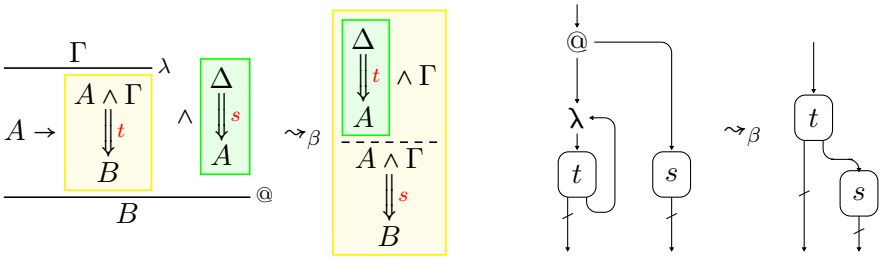


As an example, let us observe the atomic λ-term $2\,(I\,I)$ where 2 is the Church numeral $\lambda f.\lambda x.f_1\,(f_2\,x)[f_1, f_2 \leftarrow f]$ and I the identity function $\lambda x.x$. Below we display the typing judgement for this term as well as a graphical interpretation. The dotted line in the derivation is the identity inference rule and is only used to improve readability. The coloured boxes highlighted correspond to each other.





Beta reduction in the atomic λ-calculus is as expected, $(\lambda x.t)\, s \rightsquigarrow_\beta t\{s/x\}$. We use a linear substitution since there is only one occurrence of the bound variable. This reduction can be expressed as a proof rewrite rule and graph rewrite rule as shown below.

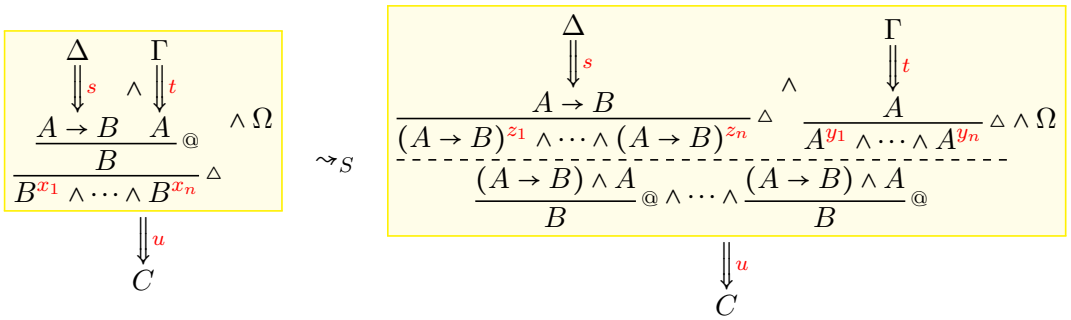


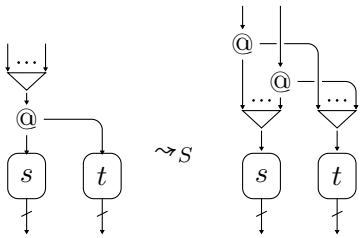
In our example $2\,(I\,I)$, after performing a β-step we see the subterm $I\,I$ become shared.

$$2\,(I\,I) = (\lambda f.\lambda x.f_1\,(f_2\,x)[f_1, f_2 \leftarrow f])\,(I\,I) \rightsquigarrow_\beta \lambda f.\lambda x.f_1\,(f_2\,x)[f_1, f_2 \leftarrow I\,I]$$

Next we consider duplication. The 'atomic' in atomic λ-calculus comes from the fact that the reduction rules allow for duplication of the constructors, i.e. we duplicate a term atomically. This method of duplication is commonly seen in graphical representations of λ-calculi with sharing [Lam90, GAL92, Mac98] and has been proven to not cause any overhead to reduction [GS17]. In our example where the term $I\,I$ is shared, the first contructor we would duplicate would be the application. Duplication of applications is handled by the rule below, where all introduced variables are fresh. The proof and graph rewritings are also shown.

$$u[x_1 \dots x_n \leftarrow s\,t] \rightsquigarrow_S u\{z_1\,y_1/x_1\} \dots \{z_n\,y_n/x_n\}[z_1 \dots z_n \leftarrow s][y_1 \dots y_n \leftarrow t]$$





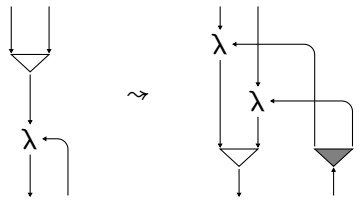
The other constructor to duplicate is abstraction. The main idea of the atomic lambda-calculus is to give a computational interpretation to the following medial rule

$$\frac{(A \vee B) \to (C \wedge D)}{(A \to C) \wedge (B \to D)}\, m$$

There are many other medial rules e.g. found in [BT01, Tiu06, Str07, GG08, AT17] and more continue to be found [PH18]. The medial rule mentioned by Brünnler and Tiu in 2001 to enable the *atomicity property* for classical logic, where contraction rules can be restricted to their atomic form. This is different from the atomic in the atomic λ-calculus, which referes to atomic duplication. This rule then allows for the following proof rewriting step when duplicating an abstraction. Note that the co-contraction rule in the derivation is morally inverted, i.e. the premise of the derivation is $A \to B$ and not $(A \vee A) \to B$.

$$\frac{A \to B}{(A \to B) \wedge (A \to B)}\, \triangle \qquad \rightsquigarrow \qquad \frac{\dfrac{A \vee A}{A} \to \dfrac{B}{B \wedge B}\, \triangle}{(A \to B) \wedge (A \to B)}\, m$$

Here, a contraction inference on a formula $A \to B$ is replaced by a medial rule, where we introduce a new inference rule shown on the antecedent of the implication above the medial (in the yellow box), which we call *co-contraction*. This rewrite can be displayed graphically



where the co-contraction is denoted as a darker version of the sharing node. The medial then corresponds to the two λ-nodes in the graph on the right taken together. However, in order to avoid introducing disjunction into our typing system, the atomic λ-calculus refines the medial rule and the co-contraction rule into a *distribution rule* (d).

$$\frac{A \to (B \wedge C)}{(A \to B) \wedge (A \to C)}\, d \qquad \sim \qquad \frac{\dfrac{A \vee A}{A} \to (B \wedge C)}{(A \to B) \wedge (A \to C)}\, m$$

With this rule comes a new construct for the term calculus, duly called the *distributor*. The distributor is the computational interpretation of the distribution rule, and is what allows

us to duplicate abstractions atomically. The full syntax of the term calculus for the atomic λ-calculus is then the grammar given before extended with

$$t[x_1, \ldots, x_n \twoheadleftarrow \lambda y\langle t_1, \ldots, t_n\rangle\overline{[\Gamma]}]$$

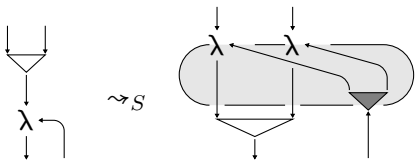
where each x_i occurs in t and becomes bound, $\langle t_1, \ldots, t_n\rangle$ is a tuple of terms, $\overline{[\Gamma]}$ is an environment: a collection of sharings and (nested) distributors that all bind variables located in the tuple of term, and y occurs within a term found in the environment. Also note that the number of terms in the tuple (n) corresponds to the number of variables bound by the distributor.

When an abstraction is to be duplicated, we introduce a distributor as opposed to duplicating the subterm as a whole. The rewrite rule is shown below, where all introduced variables are fresh.

$$u[x_1, \ldots, x_n \leftarrow \lambda y.t] \rightsquigarrow_S u[x_1, \ldots, x_n \twoheadleftarrow \lambda y\langle z_1, \ldots, z_n\rangle[z_1, \ldots, z_n \leftarrow t]]$$

The proof transformation and graph rewriting step for this reduction are shown below. This rewrite rule allows reduction to continue on the body of the abstraction which is still shared. This can be observed in both the proof and graph depictions, where a contraction inference rule is ready to proceed up the derivation of the term t. The distributor is represented by the grey box in graphical setting, leaving the sharing node to carry on with duplication.

$$\dfrac{\dfrac{\Gamma}{A \to \boxed{\begin{array}{c} A \wedge \Gamma \\ \Downarrow t \\ B \end{array}}}\lambda}{(A \to C) \wedge (A \to C)}\triangle \quad \rightsquigarrow_S \quad \dfrac{\dfrac{\Gamma}{A \to \boxed{\begin{array}{c} A \wedge \Gamma \\ \Downarrow t \\ \dfrac{B}{B \wedge B}\triangle \end{array}}}\lambda}{(A \to C) \wedge (A \to C)}d$$



As reduction continues, the variables that were initially in the tuple become substituted and replaced with terms. This can by seen in the proof normalisation as the derivations that occur underneath the contraction rule and above the distribution rule. In the graphical setting, these terms in the tuple are exactly the graphs between the distributor (grey box) and the sharing node. This continues until we are ready to eliminate the distributor, but it may be the case that we lift sharings/distributors out of the distributor first before eliminating the distributor and finish duplicating the abstraction. For example,

$$\begin{aligned} u[x_1, \ldots, x_n \twoheadleftarrow \lambda y\langle t_1, \ldots, t_n\rangle[y_1, \ldots, y_m \leftarrow y][z_1, \ldots, z_p \leftarrow s]] \rightsquigarrow_S \\ u[x_1, \ldots, x_n \twoheadleftarrow \lambda y\langle t_1, \ldots, t_n\rangle[y_1, \ldots, y_m \leftarrow y]][z_1, \ldots, z_p \leftarrow s] \end{aligned}$$

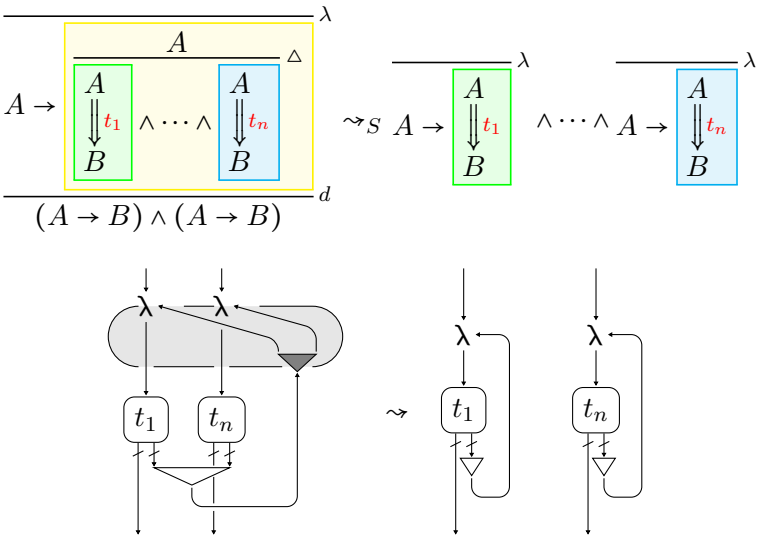
After lifting out the sharing, the distributor in the example is ready to be eliminated. Eliminating the distributor means constructing the abstractions to be substituted into u from the terms in the tuple of the distributor. The rewrite rule is shown below where we write $\vec{z}$ as a short hand notation for $z_1, \ldots, z_m$.

$$u[x_1, \ldots, x_n \twoheadleftarrow \lambda y\langle t_1, \ldots, t_n\rangle[\vec{z} \leftarrow y]] \rightsquigarrow_S u\{\lambda y_i.t_i[\vec{z}_i \leftarrow y_i]/x_i\}_{1 \le i \le n}$$

where for each $i \le n$, $\{\vec{z}_i\} = \{\vec{z}\} \cap (t_i)_{fv}$. The notation $\{t/x_i\}_{1 \le i \le n}$ is short for multiple substitutions. The abstracted y variable in the example is shared among the ts, where $\vec{z}$

contain any number of variables from each t_i. The λy is nominally also shared by the ts. What happens to that each t_i is given its own abstraction λy_i, and its own sharing $[\vec{z}_i \leftarrow y_i]$, forming $\lambda y_i.t_i[\vec{z}_i \leftarrow y_i]$ which is substituted for x_i.

The idea is a lot easier to see in proof theory and graph rewriting, where we take derivations (resp. graphs) and reorder them to obtain more abstractions.



With the reductions we've considered above, the atomic λ-calculus duplicates terms atomically which was only ever seen before in graphical calculi. This calculus also stems from the exploration of a correspondence between open deduction and the λ-calculus, and has solid foundations in proof theory. The reductions presented are proven to have the good basic, reduction property that is preserves strong normalisation with respect to the λ-calculus; discussed in Section 1.1.6. They also allow for fully lazy sharing, discussed in Section 1.1.5.

1.1.4 Scopes

The atomic λ-calculus as presented in [GHP13] has some great, natural properties: it has strong foundations in proof theory (presented in both deep inference and sequent calculus), it has an intuitive graphical interpretation, and it has nice reduction properties; PSN and full laziness. However, there is one criticism about the calculus we can make; the reduction rules are not *local*. More specifically, we can only perform the lifting rules outside of an abstraction (or distributor) if the variable bound by the abstraction does not occur in a subterm being lifted. This means an implementation of this calculus would be required to check for free variables.

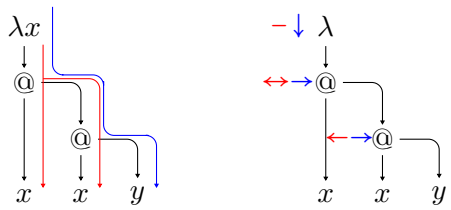
The standard solution to the problem is making variable information locally available with an explicit end-of-scope marker, as explored by Berkling and Fehr [**?**], and more recently by Hendriks and Van Oostrom [HvO03]. This allows us to identiy the skeleton of an abstraction. We use their *adbmal* ($\curlywedge$) to illustrate the idea: the constructor $\curlywedge x.N$ indicates that the subterm N does not contain occurrences of x (or that any that do occur are not available to a binder λx outside $\curlywedge x.N$). The scope of an abstraction thus

becomes explicitly indicated in the term. This opens up a distinction between *balanced* and *unbalanced* scopes: whether scopes must be properly nested, or not; for example, in $\lambda x.\lambda y.N$, a subterm $\lambda y.\lambda x.M$ is balanced, but $\lambda x.\lambda y.M$ is not. With balanced scope, one can indicate the skeleton of an abstraction; with unbalanced scope (which Hendriks and Van Oostrom dismiss) one can indicate the spine. We do so for the example term $\lambda x.\lambda y.((\lambda z.z)y)x$ below, where blue denotes the skeleton of the abstraction $\lambda.x$ and red the spine.

Balanced: $\lambda x.\lambda y.((\lambda y.(\lambda x.\lambda z.z)y)(\lambda y.x)$
Unbalanced: $\lambda x.\lambda y.(\lambda x.(\lambda y.\lambda z.z)y)(\lambda y.x)$

This constructor was used in [vOvdLZ04] only in the balanced case to define an algorithm for optimal reduction, that uses graph rewriting techniques, like Lamping's algorihtm [Lam90] which use bracket nodes to dictate enclosures, including those for abstractions. These closures update the labels on each node, that decide whether, when two sharing nodes meet, they should copy each other or eliminate each other. This is also a known as the problem of implementing (efficiently) the boxes of linear logic [Gir87].

A closely related approach is the use of *director strings*, introduced by Kennaway and Sleep in [KS88] for combinator reduction and further generalised for any reduction strategy by Fernández et al. in [FMS05b]. Director strings are an annotation on terms detailing the location of variable occurrences, providing an alternative to variable names. In the nature of substitutions, director strings would dictate the path substitutions would travel to reach the intended variables by informing at each application ($@$) if they intended variable is in the function part ($\leftarrow$), the argument part ($\rightarrow$), or even in both parts ($\leftrightarrow$). Additionally, each character in the annotation (the arrows) correspond to one variable, and for each abstraction (λ), director strings can inform us of variable bindings i.e. if the variable is bound by the abstraction ($-$) or not ($\downarrow$). For intuition, the following graphical representation of the term $\lambda x.x(xy)\{Y/y\}$ highlights the ideal paths of the substitution with annotations $-\downarrow$, $\leftrightarrow\rightarrow$, and $\leftarrow\rightarrow$.



Director strings have already been used in many implementations (see [FMS05a, FMS05b, SFM03]), and efficient reduction strategies involving director strings have already been studied in [FM99, SFM03] that involve restricting the strings to a canonical form, where these strategies are considered efficient in the sense of the number of rewrite steps (not just β-steps). However, both adbmal and director strings are more focused on the guiding of substitutions and elminating the need for α-conversion. Furthermore, although director strings are able to identify the spine of an abstraction, there is no study into extracting the spine for duplication. This would lead to heavy book-keeping steps to maintain the variable bindings between the spine and the subterms.

Previously we briefly discussed how the switch rule can identify subterms that do not contain bound variables. The main idea of this thesis comes from extending the proof system from which the atomic λ-calculus is derived with this new rule, which allows us to restrict the abstraction inference rule to an axiom, making the scope of an abstraction more flexible in the proof theory that we did not have before. This new inference changes the way we view abstractions in the typing system. Now abstractions are composed of two inference rules, the abstraction rule and the *scope rule* (switch rule).

$$\frac{\Gamma}{A \to \boxed{\begin{array}{c} A \wedge \Gamma \\ \Downarrow \\ B \end{array}}}\,\lambda \qquad \sim \qquad \frac{\overline{A \to A}^{\,\lambda} \wedge \Gamma}{A \to \boxed{\begin{array}{c} A \wedge \Gamma \\ \Downarrow \\ B \end{array}}}\,s$$

Using the previous example with director strings, we can type the term $\lambda x.x\,(x\,y)$ where x has type $A \to A$ (where we write B for short) and y has type A in the following derivation. The switch rules can then correspond to sorting the variables paths annotated with $-$ and those with $\downarrow$ for each abstraction, as the variables annotated with $\downarrow$ are exactly those 'brought out' of scope with the switch rule.

$$\frac{(B \to B \wedge B) \wedge A}{B^{\lambda x} \to \dfrac{B^{x} \wedge \boxed{\dfrac{B^{x} \wedge A^{y}}{A}\,@}}{A}\,@}\,s \qquad \frac{(B \to B \wedge B) \wedge A}{B \to (B \wedge B)^{-} \wedge A^{\downarrow}}\,s$$

The switch rule enables us to perform spine duplication. More details about this given in Section 3.3.3, but here we demonstrate the exact proof normalisation technique, using both the distribution rule of [GHP13] and the switch, that allows for spine duplication. The distribution rule is introduced when one wishes to duplicate an abstraction in the atomic λ-calculus. To demonstrate this technique, below is the proof derivation of the shared term $\lambda x.\lambda y.x\,y$ where x has type B and y has type A, and the proof derivation of the term once the distribution rule is introduced for both abstractions.

$$\frac{}{B \to \dfrac{B \wedge \overline{A \to A}^{\,\lambda}}{A \to \boxed{\dfrac{B \wedge A}{A}\,@}}\,s}\,\lambda \qquad \rightsquigarrow \qquad \frac{}{B \to \dfrac{B \wedge \overline{A \to A}^{\,\lambda}}{A \to \boxed{\dfrac{B \wedge A}{A}\,@}}\,s}\,\lambda$$

$$\frac{}{(B \to A \to A) \wedge (B \to A \to A)}\,\triangle \qquad\qquad \frac{}{(B \to A \to A) \wedge (B \to A \to A)}\,d$$

We can then proceed to duplicate the application rule as described before. Now we push the derivations underneath the distribution rule i.e. we substitute the terms into context. In the example we combine the two steps so that the derivations are pushed underneath both rules.

$$\dfrac{\dfrac{}{B \to \dfrac{\dfrac{B}{B \wedge B}\triangle \wedge A \to \dfrac{\dfrac{}{A}\lambda}{A \wedge A}\triangle}{A \to \dfrac{B \wedge A}{A}@ \wedge \dfrac{B \wedge A}{A}@}s}\lambda}{(A \to A) \wedge (A \to A)}d}{(B \to A \to A) \wedge (B \to A \to A)}d \quad \rightsquigarrow \quad \dfrac{\dfrac{}{B \to \dfrac{\dfrac{B}{B \wedge B}\triangle \wedge A \to \dfrac{\dfrac{}{A}\lambda}{A \wedge A}\triangle}{A \to (B \wedge A) \wedge (B \wedge A)}s}\lambda}{(A \to B \wedge A) \wedge (A \to B \wedge A)}d}{(B \to A \to \dfrac{B \wedge A}{A}@) \wedge (B \to A \to \dfrac{B \wedge A}{A}@)}d$$

Now between the innermost distributor and the abstraction rule introducing A is only the switch rule. Since we are duplicating an abstraction, and since we have already pushed some of the body of the abstraction into context underneath the distributor, it would make sense that these partial duplicates of abstractions have their own scope. We can therefore duplicate the switch rule accordingly. This is the core idea behind this thesis and is key to understanding what is happening. Below is the rule for when a distributor and switch interact as well as the continued example demonstrating the rule.

$$\dfrac{\dfrac{(A \to B \wedge C) \wedge D}{A \to B \wedge C \wedge D}s}{(A \to B) \wedge (A \to C \wedge D)}d \quad \rightsquigarrow \quad \dfrac{\dfrac{(A \to B \wedge C)}{(A \to B) \wedge (A \to C)}d \wedge D}{(A \to B) \wedge \dfrac{(A \to C) \wedge D}{A \to C \wedge D}s}$$

$$\dfrac{\dfrac{}{B \to \dfrac{B}{B \wedge B}\triangle \wedge \dfrac{\dfrac{}{A \to \dfrac{A}{A \wedge A}\triangle}\lambda}{(A \to A) \wedge (A \to A)}d}{\dfrac{B \wedge (A \to A)}{A \to B \wedge A}s \wedge \dfrac{B \wedge (A \to A)}{A \to B \wedge A}s}\lambda}{B \to A \to \dfrac{B \wedge A}{A}@ \wedge B \to A \to \dfrac{B \wedge A}{A}@}d$$

We then, as before, push these derivations underneath the distributor.This is something that the original atomic λ-calculus cannot do as the formulas are maintained with the switch rule.

$$\dfrac{\dfrac{}{B \to \dfrac{B}{B \wedge B}\triangle \wedge \dfrac{\dfrac{}{A \to \dfrac{A}{A \wedge A}\triangle}\lambda}{(A \to A) \wedge (A \to A)}d}{(B \wedge (A \to A)) \wedge (B \wedge (A \to A))}\lambda}{B \to \dfrac{B \wedge (A \to A)}{A \to \dfrac{B \wedge A}{A}@}s \wedge B \to \dfrac{B \wedge (A \to A)}{A \to \dfrac{B \wedge A}{A}@}s}d$$

We can now lift the innermost distributor (in the green box) outside the scope of the abstraction of B, and use a switch rule to reintroduce it.

$$\dfrac{\dfrac{\dfrac{}{B \to \dfrac{B}{B \wedge B}\,\triangle}\,\lambda \;\wedge\; \dfrac{\dfrac{}{A \to \dfrac{A}{A \wedge A}\,\triangle}\,\lambda}{(A \to A) \wedge (A \to A)}\,d}{B \to (B \wedge (A \to A)) \wedge (B \wedge (A \to A))}\,s}{B \to \dfrac{B \wedge (A \to A)}{A \to \dfrac{B \wedge A}{A}\,@}\,s \;\wedge\; B \to \dfrac{B \wedge (A \to A)}{A \to \dfrac{B \wedge A}{A}\,@}\,s}\,d$$

And just as before, we can duplicate the switch rule into the partial duplicate abstractions and push them underneath the distributor, moving the distributor closer to the abstraction.

$$\dfrac{\dfrac{\dfrac{}{B \to \dfrac{B}{B \wedge B}\,\triangle}\,\lambda}{(B \to B) \wedge (B \to B)}\,d \;\wedge\; \dfrac{\dfrac{}{A \to \dfrac{A}{A \wedge A}\,\triangle}\,\lambda}{(A \to A) \wedge (A \to A)}\,d}{\dfrac{(B \to B) \wedge (A \to A)}{B \to \dfrac{B \wedge (A \to A)}{A \to \dfrac{B \wedge A}{A}\,@}\,s}\,s \;\wedge\; \dfrac{(B \to B) \wedge (A \to A)}{B \to \dfrac{B \wedge (A \to A)}{A \to \dfrac{B \wedge A}{A}\,@}\,s}\,s}$$

Now we can eliminate any of the distributors. In the previous system without the switch, distributors as nested. In the $B \to A \to A$ in the example,the distributor for the A would have to be resolved before the distributor for the B. This corresponds to skeleton-duplication, since the formula is $B \to A \to A$ and the abstraction for A is on the spine of the abstraction for B. In the new calculus, we can resolve B first, and leave the abstraction for A as a distributor i.e. duplicated exactly where the spines of B and A coincide and no further.

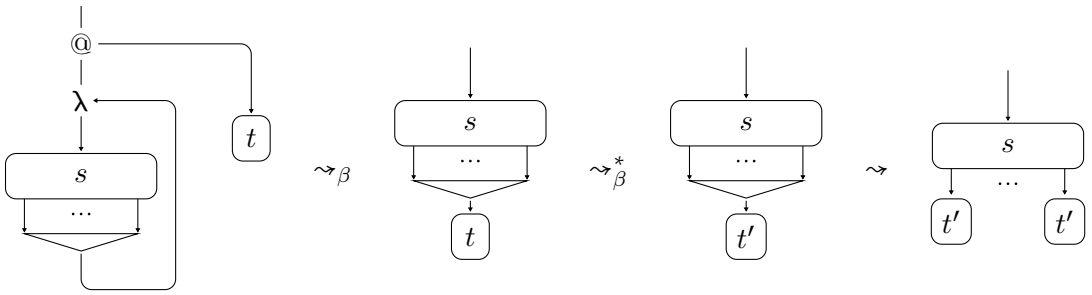
1.1.5 Duplication Strategies

Laziness

The purpose of all sharing techniques in the λ-calculus, in which multiple instances of a common subterm has one shared representation, permitting evaluation of the subterms to happen simultaneously. This desire of control is also found in explicit subtitutitions [ACCL91]. Lazy evaluation/call-by-need [AF97] postpones the duplication of a term for as long as possible, and the term is only duplicated when a value is needed during reduction e.g. if the term is the abstraction that forms part of a redex. By evaluating the term, we avoid duplicating executable work. Additionally, we can avoid evaluating the term unless it is actually required to do so i.e. until we need to duplicate it.

Sharing cannot be achieved through the use of only λ-terms: it requires technical tools. Possibly the most famous approach for expressing sharing is the use of graphs. Terms are initially encoded into a graph, and graph rewrite rules correspond to reduction in the

λ-calculus.In the picture below, observe the redex (where the @ node and λ node interact) and notice that the λ is binding multiple variables occurring in s, through the use of the sharing node (the triangle).



The picture depicts how β-reduction would look as a graph rewrite rule. In this example, the term t becomes shared after beta reduction. Then following a lazy evaluation strategy, if it is necessary to duplicate t, we can first evaluate the graph representing the term t, resulting in t', thus avoiding the repetition of work. We can then duplicate the term t' with other graph rewrite rules.

Another approach to dealing with sharing involves directly restructuring the program, called *program transformations*. When compiling the λ-calculus into the program, terms are transformed via lambda lifting [Joh85, PJ87]. Lambda lifting is a process that restructures a program, transforming local functions into global functions (from block-structured functional programs to recursive equations). Since the scope is global, it can be interpreted as sharing.

A third approach utilizes closures and memory heaps. This approach names the arguments of applications with fresh variable names, using the `let ... in ...` construct. After naming the arguments we then put the arguments into a heap such that one can look up the argument with its corresponding name. The heap then shares these arguments throughout evaluation. Laziness occurs when one needs to access the content of a variable and the corresponding expression is duplicated. Lazy evaluation is then simply memoization: when accessing the contents of a variable, the corresponding expression in the heap is first evaluated, and the heap is first updated with the evaluated expression and then the evaluated expression is duplicated. Launchbury provides the semantics of this approach to implementing lazy evaluation in [Lau93].

Full Laziness: Skeleton Duplication

The purpose of sharing is to obtain more control over the duplication of terms, which is a costly operation and may cause redundant computations. Laziness duplicates the whole expression, regardless whether the expression is evaluated due to the lazy strategy or not. However, following a lazy evaluation strategy does not mean work is not duplicated. If we look at the term taken from [Sin08], $(\lambda f.(f\,I)\,(f\,I))\,(\lambda w.(I\,I)\,w)$ where I is the identity function, after the outermost redex is reduced, following a lazy evaluation strategy the subterm $\lambda w.(I\,I)\,w$ should become shared. Regardless of the sharing technique we use, this expression is already considered a value (when considering abstractions as values which is standard), as a result the redex $I\,I$ will inescapably become duplicated.

To overcome this, and thus become more efficient in the sense that we avoid reducing terms more than once, even if they appear inside the body of an abstraction, as long as they do not mention the variable bound by the abstraction, a stronger and more powerful notion of sharing is required. The answer of course, is the appropriately named *fully lazy sharing* or again just *full laziness*. The concept of full laziness was introduced by Wadsworth in [Wad71] as a graph evaluation technique.

A *maximal free subexpression* (MFS) of an function $\lambda y.t$ is a subexpressions of t, s, such that (i) for all variables z, if $z \in (s)_{bv}$ then $z \in (\lambda y.t)_{bv}$ and if $z \in (s)_{fv}$ then $z \in (\lambda y.t)_{fv}$ and (ii) there does not exist a subexpression of t that meets the conditions of (i) and contains s as a subexpression. With this, we can say full laziness is a notion of sharing such that duplicating an abstraction expression avoids copying the maximal free subexpressions. Only the skeleton of the body of the abstraction is duplicated. The maximal free subexpressions of the term $\lambda w.(I\,I)\,w$ consist of the one subexpression $I\,I$.

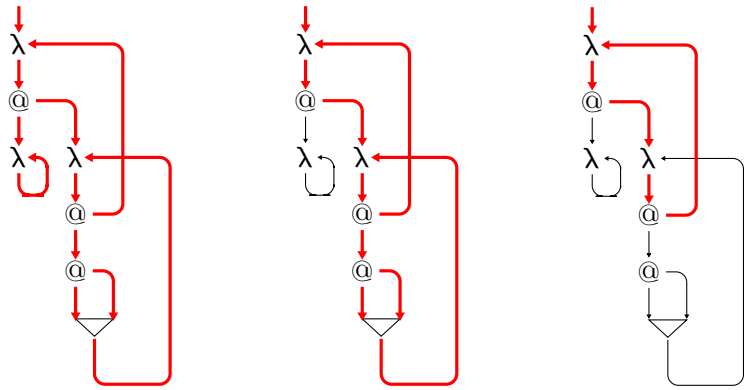
The three methods discussed for implementing sharing (graphs, closures and program transformation) and combinations of methods have been used to implement full laziness. Balabonski [Bal12] provides a nice summary of implementations that use full laziness that we summarise here. Shivers and Wand [SW04] enrich Wadsworth graph evaluation technique by representing terms as a DAG as opposed to a tree, allowing for a more simple and efficient implementation. Blanc et al. [BLM07] derive a graph implementation of fully lazy sharing by labeling the nodes of the graph. This is also done by Sinot in [Sin06] for a fully lazy graph-based abstract machine.

Full laziness has also been implemented by combining graphs with other methods. Peyton-Jones in [PJ87] discusses the transformation *fully lazy λ-lifting*: which extracts the maximal free subexpressions of an expression first before translating it into a graph.

Sestoft [Ses97] use the extraction of maximal free subexpressions from closures represented by `let` statements to outline a revision of Launchbury's operational semantics of the lazy evaluation strategy, so that it uses fully lazy sharing instead of the original lazy sharing. Sinot similarly revises Launchbury's operational semantics in [Sin08] for *completely lazy evaluation*, another notion of laziness. This is an evaluation strategy with a powerful notion of sharing. Complete laziness requires all immediate redexes in an expression to be evaluated at most once, as well as the redexes that can be generated by performing β-reductions in the body of abstractions. For example $(\lambda f.(f\,I)\,(f\,I))\,(\lambda w.(I\,I)\,w)$, a completely lazy evaluation would require the reduction sequence $(I\,I)\,w \rightsquigarrow_\beta I\,w \rightsquigarrow_\beta w$. This is clearly stronger than the requirements of lazy evaluation as implemented in [PJ87], which would consider $\lambda w.(I\,I)\,w$ a value and does not require the redexes underneath the abstraction to be evaluated. Sinot claims that complete laziness has more 'sharing power' than full laziness. This statement is also claimed explicitly in [HG91]. Here he states that complete laziness captures the spirit of full laziness, and that some implementations [SW04, Sin08] that are known to implement full laziness that are likely to follow the semantics of complete laziness, because they use only graph rewriting techniques where all immediate redexes are reduced. This remains unchecked however.

Full Laziness: Spine Duplication

Expanding on the idea of full laziness is *spine duplication*. The concept of this notion of sharing is easy to see when considering expressions as graphs. When an abstraction needs to be duplicated (when a copy is required), we duplicate the shortest path possible to the variables bound to the abstraction from the root node (the spine). The diagram underneath illustrates this, where the thick red line indicates the nodes that will get duplicated with each duplication strategy mentioned so far.



These degrees of sharing have all been seen before e.g. laziness used in to define its big step operational semtantics by Launchbury in [Lau93], full laziness used in the abstract machine defined in [Sin06], and spine duplication is implemented with labels in [BLM07]. The idea of using labels when sharing terms is fairly common, and can have some great advantages. One notable use of labels for sharings is Lamping's algorithm [Lam90] for optimal reduction in the sense of Lévy [Lév80].

1.1.6 Preservation of Strong Normalisation

In a desire to capture the behaviour of computation in its use of computational resources, *explicit-substitution calculi* [ACCL91] were introduced to obtain more control over the substitution process. This is done by representing substitutions explicitly in the term calculus i.e. we implement substitution based of its mathematical recursive definition. Abadi et al. introduced in [ACCL91] the $\lambda\sigma$-calculus as a bridge between the classical λ-calculus and more concrete implementations of the calculus, which used explicit substitutions that could be delayed and stored. This can be seen as an approach to obtaining control over duplication of terms in term-calculi, as compared to the graph reductions.

Given a calculus with explicit substitutions, we say that it *preserves β-strong normalisation* (PSN) if for a term $M \in \lambda$ is strongly normalising then its compilation into the calculus with explicit substitutions is strongly normalising i.e. no infinite reductions are created by the use of explicit substitutions. PSN may seem like a natural property, but it was shown by Mellies in [Mel95] that the $\lambda\sigma$-calculus may not terminate. Since then, PSN has become increasingly studied. To our knowledge, there exist three approaches to proving strong normalisation with respect to the λ-calculus.

One method works by defining a typing system that uses *intersection types* [CDC78, CDC80]. By providing an intersection type discipline for the terms of the calculus, we verify the the typed terms are strongly normalising. Then, if there is a correspondence between the calculus and the typing system with the λ-calculus, the property of preserving strong normalisation (PSN) is immediate. This approach was taken by Kesner and Ó Conchúir in [KC08]. A deep inference system for intersection types has been explored by Paulus and Heijltjes in [PH18].

Another way of proving PSN is by proving the calculus holds the *IE property* (relating termination of **I**mplicit substitutions to termination of **E**xplicit substitutions) which states that for any given terms u and t of the calculus and variable x, the two facts that strong normalisation of the term $t\{u/x\}$ (where $\{u/x\}$ denotes implicit substitution) and strong normalisation of the term u, imply strong normalisation of the term $t[u/x]$, where $[u/x]$ denotes an explicit substitution. Intuitively, this property is saying explicit substitutions implement implicit substitutions and nothing more than that. This technique of proving PSN was first introduced by Kesner in [Kes08] and further explored in [Kes09]. This method was used by Kesner and Accattoli in [AK12b].

The third method, and the method we choose, translates the calculus into a "weakened" calculus which is not able to erase terms. David and Guillaume introduced a calculus that uses explicit substitutions and *labels* which corresponded to explicit weakenings [DG01] in order to address the problem raised by Mellies' counter-example, and showed this calculus has PSN and controlled composition of explicit substitutions i.e. converting explicit substitutions into implicit substitutions. Then Cosmo, Kesner and Polonovski show in [DCKP00] that typed terms in this calculus are strongly normalising by using variable names instead of de Bruijn indices. They achieve this by presenting a variant of the calculus in [DG01] that makes use of Linear Logic's proof-nets (a graphical representation), suggesting the weakening and contraction (erasure and duplication) can be added to the calculus without loss of termination. The correspondence between the proof-nets and the weakening calculus is made more clear in [KL05]. The weakening calculus relates to Barendregts λ-I calculus [Bar84] (a relevant λ-calculus, where variables must occur). It originates from Klop's version with memory in [Klo80].

The approach of proving PSN by reducing the problem to proving PSN for the weakening calculus was done in [AK12a], where they reduced the problem to a second calculus which is used as an auxiliary tool to show confluence for the original calculus preserving infinite reductions. PSN was then reduced to an already proven result found in [AK12b]. The atomic λ-calculus was reduced to a weakening calculus presented in the same paper [GHP13].

1.2 Curry-Howard Correspondence

The Curry-Howard correspondence relates computer programming and mathematical logic, where formulas are types, proofs are programs, and proof normalisation is computation. This correspondence allows us to directly interpret results in proof theory as results about computation.

Curry observed that the types of combinators could be observed as axiom schemes for intuitionistic implicational logic (1934) and later in Hilbert systems (1958). Howard

observed (1969) another correspondence between proof systems and computation: natural deduction and the simply typed λ-calculus. One can use the following rules to derive a proof in minimal logic that corresponds to a term in the simply typed λ-calculus.

$$\frac{}{A \vdash x : A}\,\text{VAR} \qquad \frac{\Gamma, A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\,\text{ABS} \qquad \frac{\Gamma \vdash t : A \to B \quad \Delta \vdash u : A}{\Gamma, \Delta \vdash t\, u : B}\,\text{APP}$$

These proofs are in the style of natural deduction, and can translated into the style of open deduction, as discussed in [GHP13]. The atomic λ-calculus was *derived* from deep inference (in particular the medial rule) as a Curry-Howard interpretation. Another common rule used in deep inference is the switch rule, so this leaves the question: what is the Curry-Howard interpretation of the switch rule, in computation, and how does it interact with the distributor, the computational interpretation of the medial rule.

$$\frac{\dfrac{A \vee A}{A} \to \dfrac{B}{B \wedge B}\,\triangle}{(A \to B) \wedge (A \to B)}\,m \qquad \frac{(A \to B) \wedge C}{A \to (B \wedge C)}\,s$$

1.3 A calculus that achieves spine duplication

The atomic λ-calculus [GHP13] makes a good theoretical foundations for understanding the full laziness. This is due to its strong connections with proof theory, a close intuition with graph rewriting, and powerful reductions that allow for full laziness with good, natural properties such as PSN.

We present a calculus that can naturally perform a more powerful notion of full laziness. We maintain the strong connection with proof theory, providing a Curry-Howard correspondence between logic and this calculus. More concretely, we describe the correspondence between the switch rule commonly seen in deep inference and scopes in programming. We provide a natural graphical intuition, as well as proving the reductions of this calculus satisfy PSN as well as allowing for spine duplication. Moreover, we provide a calculus which implements duplication through the use of environments and as far has only been witnessed to use graphs and labels [BLM07, Bal12].

Chapter 2

Switch and End-Of-Scope

言为心声
yán wéi xīnshēng
Words are the voice of the mind

In this chapter we explore the relationship between a switch rule in deep inference and explicit end-of-scope constructors. An end-of-scope constructor helps determine the binding of variables to abstractions in the λ-calculus. As an example, take de Bruijn's indices [dB72], where variables are represented by natural numbers that denote the number of binders in scope between the variable and its correpsonding binder. A term is displayed below in the λ-calculus and the equivalent using de Bruijn's notation.

$$\lambda x.\lambda y.\lambda z.x\,z\,(y\,z) \qquad\qquad \lambda.\lambda.\lambda.2\,0\,(1\,0)$$

The idea of scope can be made more explicit by representing natural numbers $\mathcal{N}$ by the following recursive definition

$$\mathcal{N} \quad ::= \quad 0 \quad | \quad S\mathcal{N}$$

where a natural number n are represented by n applications of the successor function on the number 0. This idea of typing these terms was found in [FPT99, vOvdLZ04], and so we follow their exposition. These successor functions S can then be viewed as an explicit end-of-scope constructor in the term calculus, that dictate the scope between the occurrence of a variable (0) and its corresponding binder (λ). We can express the previous example as

$$\lambda.\lambda.\lambda.(SS0)\,0\,((S0)\,0)$$

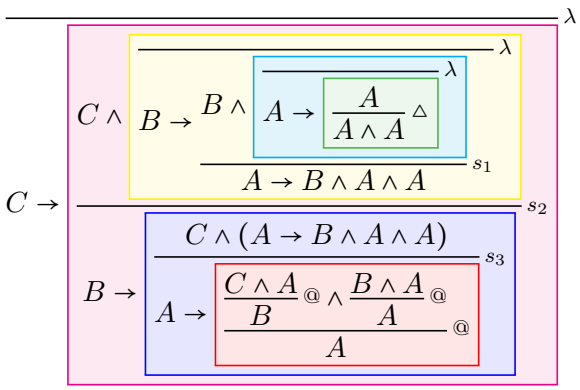
Making the scope more explicit allows for *scope manipulation*; liberating the S function from applying only to numbers to becoming a term constructor in the calculus. For example, consider the situation where we have the term $\lambda.\lambda.(S0)\,(S0)$ where we have two calls of the successor function. In this term, the application is in the scope of the second (innermost) abstraction, and the variables are not. It is possible to rewrite this term in a way that maintains bindings but alters the scope so that the application is not in scope of the innermost abstraction i.e. $\rightsquigarrow \lambda.\lambda.S(0\,0)$. The idea that switch is a proof theoretic equivalent to the term constructor S is fundamental to our work, and we explore what

scope manipulation means for the atomic λ-calculus, discovering new reduction techniques that were otherwise not possible.

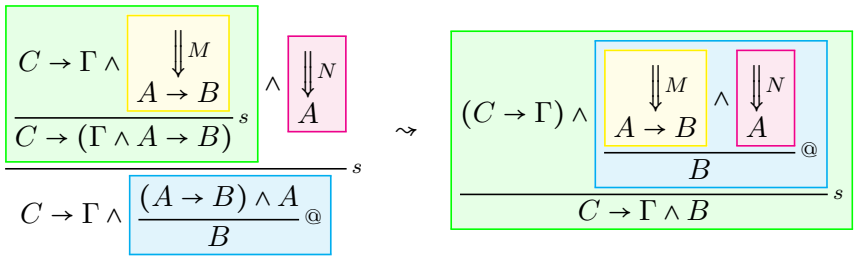
Let us consider the open deduction used for the λ-calculus. We represent the successor function in a term with the following switch rule

$$\frac{(A \to B) \wedge C}{A \to (B \wedge C)}\, s$$

The intuition behind the rule is as follows: $A \to B$ is the type of a function $\lambda.N$, and C is the type of a variable that is not captured by the λ. The switch rule 'brings' the variable into context i.e. we know it will occur in the body of the abstraction (N). For a λ-term M in the de Bruijn notation using 0 and S, we can express its typing derivation in open deduction in the same way as before, where we restrict the abstraction rule to an axiom and use the switch rule for each occurrence of the successor function in the term. For example, $\lambda.\lambda.\lambda.(S_2 S_3 0)\, 0\, ((S_1 0)\, 0)$ has the typing derivation below, where we identify each successor function and switch rule such that $S_i = s_i$, and where $B = A \to A$ and $C = A \to A \to A$.

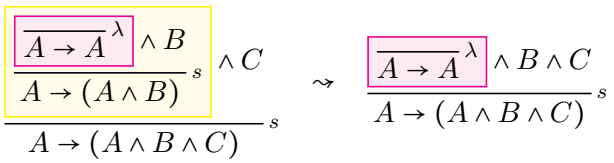


In the sense of proof theory, scope manipulation corresponds to the rewrite rules that involve the switch rule. Considering the term rewrite rule discussed before, $(SM)\,SN \rightsquigarrow S(M\,N)$, where M and N are subterms, the equivalent proof rewrite rule is,



The main result of this work is studying the effects of adding the switch rule to the typing system, introducing the *spinal atomic λ-calculus* (Λ_a^S). By making the scope explicit in the typing system, we unlock a new reduction scheme that allows for spine duplication. So far, we have looked at the proof theory from the perspective of Be Bruijn's calculus where we make the scopes explicit. Now we explore the λ-calculus from the perspective of the proof theory.

We first notice that we can generalise the switch rule so that it brings multiple variables into the scope of an abstraction. This allows for the following rewrite rule.



As a result of this rule, we can rewrite proofs such that the maximum number of switch rules in a derivation of a term is the number of abstraction rules. In the rest of this document we discuss the calculus with explicit sharing interpreted from the proof theory using the switch rule, and the proof normalisation technique that corresponds to duplication of shared terms and how the switch rule affect duplication.

Chapter 3

Spinal atomic λ-calculus

水滴石穿
shuǐdī shíchuān
Dripping water wears through rock
- Persistence pays off

This chapter introduces a refinement of the atomic λ-calculus. Our aim is to obtain a calculus that can naturally perform spine duplication by discovering the Curry-Howard correspondence to the switch rule in open deduction. We extend the typing system used with the switch rule, and define the terms based of the derivations.

This chapter starts by presenting the *pre-terms* of Λ_a^S, then defining the *terms*. We show the translations to and from the λ-calculus, describing the relationship. We describe the reduction rules, including β-reduction, where we describe how duplication is handled by this calculus and how it differs from that of the original. Lastly, we discuss the typing system used for our calculus, which is an extension from the old, and show subject reduction: the type of a term is preserved during reduction. We also explain why the addition of this switch rule to the typing system is necessary for spine duplication, and why the original typing system was not able to allow this strategy of duplication.

Before we formally introduce the spinal atomic λ-calculus, we provide some intuition. First we show how we will represent the scope of an abstraction in our calculus. We represent abstractions with the syntax $x\langle\, x\,\rangle.t$, where t is a term and x is the variable located free in t that is captured. Figure 3.1 shows the typing derivation for this abstraction. The switch rule is used to 'bring into context' the free variables of t that are not captured by the abstraction (the Δ).

$$\dfrac{\dfrac{\top}{A \to A}\,\lambda \wedge \Delta}{A^x \to \boxed{\begin{array}{c} A^x \wedge \Delta \\ \Downarrow t \\ C \end{array}}}\,s$$

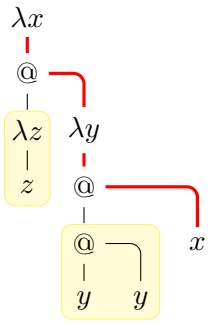
Figure 3.1: The typing derivation for $x\langle\, x\,\rangle.t$

This work introduces the notion of a *phantom-abstraction*. These can be thought of as partially duplicated abstractions; the abstraction has been duplicated where the bound variable has not yet been duplicated. The syntax for these is $d\langle\, x_1, \ldots, x_n \,\rangle.t$, where we can abbreviate the variables in the tuple as $\vec{x}$. Figure 3.2 shows the typing derivation for this phantom-abstraction. The d is the *phantom-variable*, which will eventually be the binding variable once duplication of the abstraction is complete. The variables in the tuple $\vec{x}$ are the variables in t that will be substituted for during duplication.

$$\dfrac{(A \to \Gamma) \wedge \Delta}{A^d \to \boxed{\begin{array}{c}\Gamma^{\vec{x}} \wedge \Delta \\ \Downarrow t \\ C\end{array}}}\, s$$

Figure 3.2: The typing derivation for $d\langle\, \vec{x} \,\rangle.t$

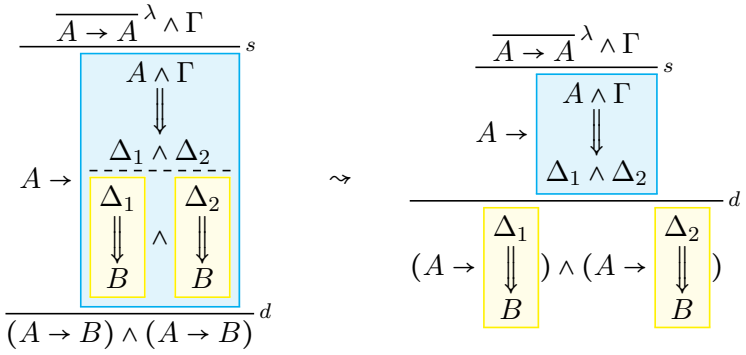
When considering the spine as discussed in Chapter 1, the variables in the tuple are exactly those connected to the spine that represent a subterm with a free occurrence of the binding variable. We repeat here the same example on the first page of the introduction. For the spine of λx here to be duplicated, the abstraction λy would be duplicated and not the subterm $y\,y$. Therefore, the intuition behind the phantom-abstraction is that we would have a partial duplicate of the λy, i.e. $y\langle\, w \,\rangle.w\,x$ where w is a fresh variable that represents the subterm $y\,y$. The notion of phantom-abstraction is then important here to achieve spine duplication.



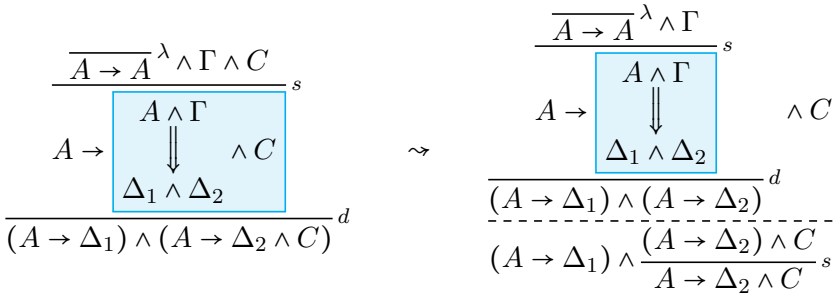
This calculus will introduce phantom-abstractions naturally when duplicating an abstraction. Similarly as described in Section 1.1.3, abstractions are duplicated with the distributor construct. This is shown below in the proof theory, which is the almost the same as in [GHP13] (because of the switch rule).

$$\dfrac{\dfrac{\overline{A \to A}^{\,\lambda} \wedge \Gamma}{A \to \boxed{\begin{array}{c}A \wedge \Gamma \\ \Downarrow \\ B\end{array}}}\, s}{(A \to B) \wedge (A \to B)}\, \triangle \quad \rightsquigarrow \quad \dfrac{\dfrac{\overline{A \to A}^{\,\lambda} \wedge \Gamma}{A \to \boxed{\begin{array}{c}A \wedge \Gamma \\ \Downarrow \\ \dfrac{B}{B \wedge B}\,\triangle\end{array}}}\, s}{(A \to B) \wedge (A \to B)}\, d$$

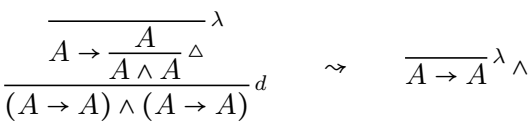
We know use some proof rewriting rules that were not used before. As the sharing proceeds through the derivation creating multiple (in the example, two) derivations, we can *flush* the derivations underneath the distributor. The formulars $A \to B$ underneath the distributor represent the phantom-abstractions, and this flushing (as displayed below) represent substituting terms into the variables that are named inside the tuple of the phantom-abstraction. As an example, before we could have $d\langle\, w \,\rangle.t$ where d has type A and w has type B, and afterwards we could have $d\langle\, \vec{x} \,\rangle.t\{s/w\}$ where $\vec{x} = (s)_{fv}$, and that each variable in $\vec{x}$ has a type listed in Δ_1.



The switch rule here becomes vital. We can 'declutter' the distributor by removing the types of variables that we know are free in the distributor and are not captured by the abstraction, just like C in the derivation below. We can instead introduce a switch rule for the phantom-abstraction underneath the distributor, to bring C into context of the term independently of the distributor. This would mean that, if we know that for variable x in $d\langle\, \vec{x} \cdot x \,\rangle.t$ corresponds to a term that does not have the binding variable occurring free inside of it, we can remove it from the list of tuples.



Eliminating the distributor is now simpler than before because all the derivations have either been flushed or decluttered. An example of this is shown below. In the term calculus, eliminating a distributor results in converting the phantom-abstractions into abstractions.



3.1 Calculus

3.1.1 Pre-Terms and Terms

We formally introduce the syntax of the *spinal atomic λ-calculus* (Λ_a^S) pre-terms and terms.

Definition 1 (Pre-Terms). *The* pre-terms $t \in \Lambda_a^S$ *are defined by the following syntax*

$$t \quad ::= \quad x \quad | \quad t\,t \quad | \quad x\langle\, y_1, \ldots, y_n \,\rangle.t \quad | \quad t[\Gamma]$$

$$[\Gamma] \quad ::= \quad [x_1, \ldots, x_n \leftarrow t] \quad | \quad [e_1\langle\, x_1^1 \ldots x_{k_1}^1 \,\rangle \ldots e_n\langle\, x_1^n \ldots x_{k_n}^n \,\rangle \,|\, d\langle\, \vec{y} \,\rangle\, \overline{[\Gamma]}]$$

$$\overline{[\Gamma]} \quad ::= \quad [\Gamma] \quad | \quad \overline{[\Gamma]}[\Gamma]$$

Where we use the following variable and naming conventions

- w, x, y, z are *variables*
- c, d, e, f, g are *phantom-variables*
- r, s, t, u are *(pre-)terms*
- $s\,t$ is an *application*
- $c\langle\, x_1, \ldots, x_n \,\rangle.t$ is a *phantom-abstraction*
- $x\langle\, x \,\rangle.t$ is an *abstraction*
- $c\langle\, x_1, \ldots, x_n \,\rangle$ is a *cover*
- $[\Gamma]$ is a *closure*
- $\overline{[\Gamma]}$ is an *environment*
- $[x_1, \ldots, x_n \leftarrow t]$ is a *sharing*
- $[\leftarrow s]$ is a *weakening*
- $[e_1\langle\, x_1^1, \ldots, x_{k_1}^1 \,\rangle \ldots e_n\langle\, x_1^n, \ldots, x_{k_n}^n \,\rangle \,|\, c\langle\, y_1, \ldots, y_m \,\rangle\, \overline{[\Gamma]}]$ is a *distributor*

Additionally, we sometimes shorten lists of variables as $\vec{x} = x_1, \ldots, x_n$. Note that an abstraction and a phantom-abstraction are two instances of the same construct, the same can be said for sharing and weakening.

Terms will be the preterms in Λ_a^S where variables and phantom-variables are correctly and sensibly bound. In order to define this, it is essential to define the free and bound variables, phantom-variables and covers of pre-terms. We then use these definitions to further define the constraints on pre-terms that result in terms for the spinal atomic λ-calculus.

Definition 2 (Free and Bound Variables). *The free variables* $(-)_{fv}$ *and bound variables* $(-)_{bv}$ *of a pre-term* t *is defined as follows*

$$\begin{aligned}
(x)_{fv} &= \{x\} & (x)_{bv} &= \{\} \\
(s\,t)_{fv} &= (s)_{fv} \cup (t)_{fv} & (s\,t)_{bv} &= (s)_{bv} \cup (t)_{bv} \\
(x\langle\, x \,\rangle.t)_{fv} &= (t)_{fv} - \{x\} & (x\langle\, x \,\rangle.t)_{bv} &= (t)_{bv} \cup \{x\} \\
(c\langle\, \vec{x} \,\rangle.t)_{fv} &= (t)_{fv} & (c\langle\, \vec{x} \,\rangle.t)_{bv} &= (t)_{bv} \\
(u[\vec{x} \leftarrow t])_{fv} &= (u)_{fv} \cup (t)_{fv} - \{\vec{x}\} & (u[\vec{x} \leftarrow t])_{bv} &= (u)_{bv} \cup (t)_{bv} \cup \{\vec{x}\}
\end{aligned}$$

$$(u[e_1\langle\, x_1^1,\ldots,x_{k_1}^1\,\rangle\ldots e_n\langle\, x_1^n,\ldots,x_{k_n}^n\,\rangle \mid c\langle\, c\,\rangle\,\overline{[\Gamma]}])_{fv} = (u\overline{[\Gamma]})_{fv} - \{c\}$$
$$(u[e_1\langle\, x_1^1,\ldots,x_{k_1}^1\,\rangle\ldots e_n\langle\, x_1^n,\ldots,x_{k_n}^n\,\rangle \mid c\langle\, c\,\rangle\,\overline{[\Gamma]}])_{bv} = (u\overline{[\Gamma]})_{bv} \cup \{c\}$$

$$(u[e_1\langle\, x_1^1,\ldots,x_{k_1}^1\,\rangle\ldots e_n\langle\, x_1^n,\ldots,x_{k_n}^n\,\rangle \mid c\langle\, \vec{y}\,\rangle\,\overline{[\Gamma]}])_{fv} = (u\overline{[\Gamma]})_{fv}$$
$$(u[e_1\langle\, x_1^1,\ldots,x_{k_1}^1\,\rangle\ldots e_n\langle\, x_1^n,\ldots,x_{k_n}^n\,\rangle \mid c\langle\, \vec{y}\,\rangle\,\overline{[\Gamma]}])_{bv} = (u\overline{[\Gamma]})_{bv}$$

As an example, take the pre-term

$$t = e_2\langle\, w_2\,\rangle.w_2\,((e_1\langle\, w_1\,\rangle.w_1)\,z)[e_1\langle\, w_1\,\rangle, e_2\langle\, w_2\,\rangle \mid c\langle\, y\,\rangle\,[w_1, w_2 \leftarrow x\langle\, x\,\rangle.x\,y]]$$

$$(t)_{fv} = \{y, z\} \qquad (t)_{bv} = \{w_1, w_2, x\}$$

Note: The distributor $u[e_1\langle\, x_1^1,\ldots,x_{k_1}^1\,\rangle\ldots e_n\langle\, x_1^n,\ldots,x_{k_n}^n\,\rangle \mid c\langle\, \vec{y}\,\rangle\,\overline{[\Gamma]}]$ does not bind variables x_j^i for $1 \le i \le n$ and $1 \le j \le k_i$. The same is true for the variables $\vec{y}$ in the phantom-abstraction $x\langle\, \vec{y}\,\rangle.t$.

Definition 3 (Free and Bound Phanton-Variables)**.** *The free phantom-variables* $(-)_{fp}$ *and bound phantom-variables* $(-)_{bp}$ *of the pre-term* t *is defined as follows*

$$\begin{aligned}
(x)_{fp} &= \{\} & (x)_{bp} &= \{\}\\
(s\,t)_{fp} &= (s)_{fp} \cup (t)_{fp} & (s\,t)_{bp} &= (s)_{bp} \cup (t)_{bp}\\
(x\langle\, x\,\rangle.t)_{fp} &= (t)_{fp} & (x\langle\, x\,\rangle.t)_{bp} &= (t)_{bp}\\
(c\langle\, \vec{x}\,\rangle.t)_{fp} &= (t)_{fp} \cup \{c\} & (c\langle\, \vec{x}\,\rangle.t)_{bp} &= (t)_{bp}\\
(u[\vec{x} \leftarrow t])_{fp} &= (u)_{fp} \cup (t)_{fp} & (u[\vec{x} \leftarrow t])_{bp} &= (u)_{bp} \cup (t)_{bp}
\end{aligned}$$

$$(u[e_1\langle\, x_1^1,\ldots,x_{k_1}^1\,\rangle\ldots e_n\langle\, x_1^n,\ldots,x_{k_n}^n\,\rangle \mid c\langle\, c\,\rangle\,\overline{[\Gamma]}])_{fp} = (u\overline{[\Gamma]})_{fp} - \{e_1,\ldots,e_n\}$$
$$(u[e_1\langle\, x_1^1,\ldots,x_{k_1}^1\,\rangle\ldots e_n\langle\, x_1^n,\ldots,x_{k_n}^n\,\rangle \mid c\langle\, c\,\rangle\,\overline{[\Gamma]}])_{bp} = (u\overline{[\Gamma]})_{bp} \cup \{e_1,\ldots,e_n\}$$

$$(u[e_1\langle\, x_1^1,\ldots,x_{k_1}^1\,\rangle\ldots e_n\langle\, x_1^n,\ldots,x_{k_n}^n\,\rangle \mid c\langle\, \vec{y}\,\rangle\,\overline{[\Gamma]}])_{fp} = (u\overline{[\Gamma]})_{fp} \cup \{c\} - \{e_1,\ldots,e_n\}$$
$$(u[e_1\langle\, x_1^1,\ldots,x_{k_1}^1\,\rangle\ldots e_n\langle\, x_1^n,\ldots,x_{k_n}^n\,\rangle \mid c\langle\, \vec{y}\,\rangle\,\overline{[\Gamma]}])_{bp} = (u\overline{[\Gamma]})_{bp} \cup \{e_1,\ldots,e_n\}$$

As an example, for the pre-term above

$$(t)_{fp} = \{c\} \qquad (t)_{bp} = \{e_1, e_2\}$$

Definition 4 (Free and Bound Covers). *The free covers* $(-)_{fc}$ *and bound covers* $(-)_{bc}$ *of a pre-term* t *is defined as follows*

$$\begin{aligned}
(x)_{fc} &= \{\} & (x)_{bc} &= \{\} \\
(s\,t)_{fc} &= (s)_{fc} \cup (t)_{fc} & (s\,t)_{bc} &= (s)_{bc} \cup (t)_{bc} \\
(x\langle\, x \,\rangle.t)_{fc} &= (t)_{fc} & (x\langle\, x \,\rangle.t)_{bc} &= (t)_{bc} \\
(c\langle\, \vec{x} \,\rangle.t)_{fc} &= (t)_{fc} \cup \{c\langle\, \vec{x} \,\rangle\} & (c\langle\, \vec{x} \,\rangle.t)_{bc} &= (t)_{bc} \\
(u[\vec{x} \leftarrow t])_{fc} &= (u)_{fc} \cup (t)_{fc} & (u[\vec{x} \leftarrow t])_{bc} &= (u)_{bc} \cup (t)_{bc}
\end{aligned}$$

$$(u[e_1\langle\, x_1^1, \ldots, x_{k_1}^1 \,\rangle \ldots e_n\langle\, x_1^n, \ldots, x_{k_n}^n \,\rangle \,|\, c\langle\, c \,\rangle\, \overline{[\Gamma]}])_{fc} = (u\overline{[\Gamma]})_{fc} - \{e_i\langle\, x_1^i \ldots x_{k_i}^i \,\rangle\}_{i \le n}$$
$$(u[e_1\langle\, x_1^1, \ldots, x_{k_1}^1 \,\rangle \ldots e_n\langle\, x_1^n, \ldots, x_{k_n}^n \,\rangle \,|\, c\langle\, c \,\rangle\, \overline{[\Gamma]}])_{bc} = (u\overline{[\Gamma]})_{bc} \cup \{e_i\langle\, x_1^i \ldots x_{k_i}^i \,\rangle\}_{i \le n}$$

$$(u[e_1\langle\, x_1^1, \ldots, x_{k_1}^1 \,\rangle \ldots e_n\langle\, x_1^n, \ldots, x_{k_n}^n \,\rangle \,|\, c\langle\, \vec{y} \,\rangle\, \overline{[\Gamma]}])_{fc} = (u\overline{[\Gamma]})_{fc} \cup \{c\langle\, \vec{y} \,\rangle\} - \{e_i\langle\, x_1^i \ldots x_{k_i}^i \,\rangle\}_{i \le n}$$
$$(u[e_1\langle\, x_1^1, \ldots, x_{k_1}^1 \,\rangle \ldots e_n\langle\, x_1^n, \ldots, x_{k_n}^n \,\rangle \,|\, c\langle\, \vec{y} \,\rangle\, \overline{[\Gamma]}])_{bc} = (u\overline{[\Gamma]})_{bc} \cup \{e_i\langle\, x_1^i \ldots x_{k_i}^i \,\rangle\}_{i \le n}$$

As an example, for the pre-term before

$$(t)_{fc} = \{c\langle\, y \,\rangle\} \qquad (t)_{bc} = \{e_1\langle\, w_1 \,\rangle, e_2\langle\, w_2 \,\rangle\}$$

We now use these definitions to precisely define the Λ_a^S-terms.

Definition 5 (Terms). *A* term $t \in \Lambda_a^S$ *is a pre-term with the following constraints*

1. *Each variable may occur at most once.*
2. *In an abstraction* $x\langle\, x \,\rangle.t$, $x \in (t)_{fv}$.
3. *In a phantom-abstraction* $c\langle\, x_1, \ldots, x_n \,\rangle.t$, $\{x_1, \ldots, x_n\} \subset (t)_{fv}$.
4. *In a sharing* $u[x_1, \ldots, x_n \leftarrow t]$, $\{x_1, \ldots, x_n\} \subset (u)_{fv}$.
5. *In a distributor* $u[e_1\langle\, w_1^1, \ldots, w_{k_1}^1 \,\rangle \ldots e_n\langle\, w_1^n, \ldots, w_{k_n}^n \,\rangle \,|\, c\langle\, c \,\rangle\, \overline{[\Gamma]}]$
   (a) *For all* $1 \le i \le n$ *and* $1 \le m \le k_n$, $w_m^i (u)_{fv}$ *and becomes bound by* $\overline{[\Gamma]}$.
   (b) $\{e_1\langle\, w_1^1, \ldots, w_{k_1}^1 \,\rangle, \ldots, e_n\langle\, w_1^n, \ldots, w_{k_n}^n \,\rangle\} \subset (u)_{fc}$, *and* $\{e_1, \ldots, e_n\} \subset (u)_{fp}$, *and each* e_i *becomes bound.*
   (c) *The variable* c *occurs somewhere in the environments* $\overline{[\Gamma]}$.
6. *In a distributor* $u[e_1\langle\, w_1^1, \ldots, w_{k_1}^1 \,\rangle \ldots e_n\langle\, w_1^n, \ldots, w_{k_n}^n \,\rangle \,|\, c\langle\, y_1, \ldots, y_m \,\rangle\, \overline{[\Gamma]}]$
   (a) *Both* $5(a)$ *and* $5(b)$ *hold.*
   (b) *For all* $1 \le i \le m$, y_i *occurs in the environments* $\overline{[\Gamma]}$.

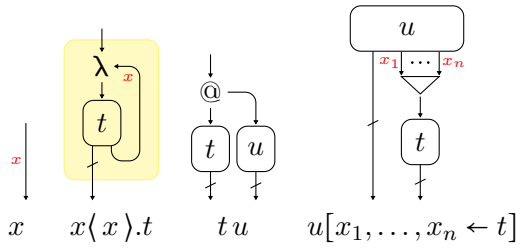


Figure 3.3: Graphical interpretation of some Λ_a^S terms

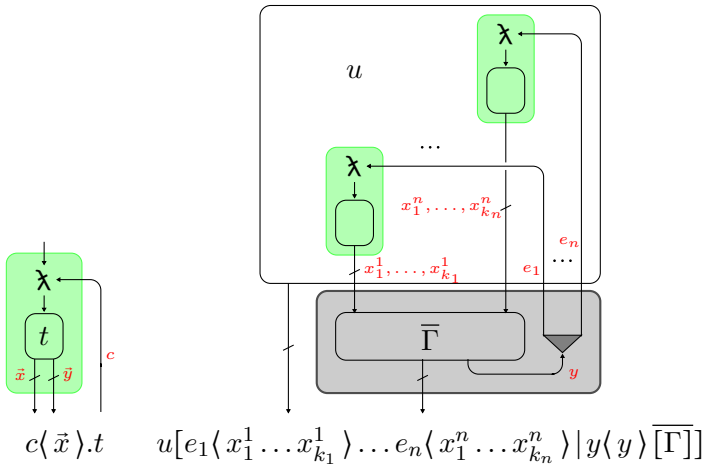
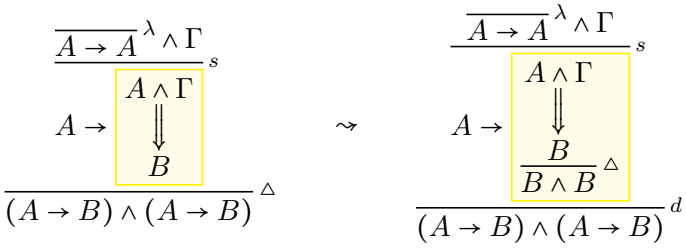


Figure 3.4: Graphical interpretation of phantom-abstraction and the distributor

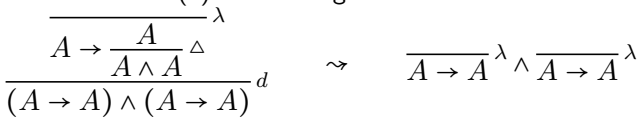
Figure 3.3 shows the graphical interpretation of some Λ_a^S-terms, that is variable, abstraction, application, and sharing. The nodes used in the graphs are the abstraction node (λ), application node (@) and a sharing node ($\triangle$). We annotate the scope of abstractions with a yellow box. Highlighting the scopes allow us to see the locations of sharings in graphical format i.e. we can distinguish between $x\langle\, x \,\rangle.t[y_1, y_2 \leftarrow s]$ and $(x\langle\, x \,\rangle.t)[y_1, y_2 \leftarrow s]$. These interpretations are an adaptation of those used in the original paper [GHP13].

The spinal atomic λ-calculus introduces 2 new constructors, the phantom-abstraction and the refined distributor. Figure 3.4 shows the graphical interpretation of these constructors. The phantom-abstraction node is similar to the abstraction node, but the input wire comes from a term out of scope. We annotate the scopes of phantom-abstractions with a green box. The free variables of the body of the phantom-abstraction are one of two categories: either they are captured by a sharing located inside a distributor that binds the phantom-variable, in which case the variable is named in the brackets, or they are not and they are not named. The graph for $c\langle\, \vec{x} \,\rangle.t$ shows the variables for both $\vec{x}$ and $\vec{y}$ where $\{\vec{y}\} = (t)_{fv} - \{\vec{x}\}$. The variables $\vec{x}$ are, intuitively, those that will be captured by the environment in the distributor that captures the phantom-variable c.

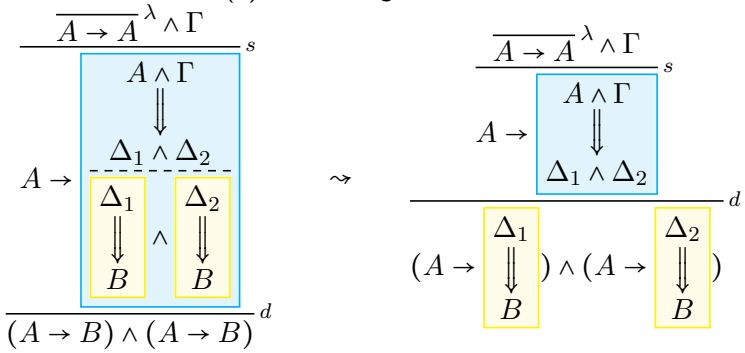
The distributor makes use of a cosharing node, which acts similar as a sharing but for wires in the opposite direction. The distributor in this calculus differs from the original in



(a) Introducing Distributor



(b) Eliminating Distributor



(c) Flushing terms

Figure 3.5: The distributor in open deduction

[GHP13] since in the original calculus, the distributor stores terms in a tuple and substitutes the terms into the body when the distributor is ready to be eliminated whereas our distributor substitutes the terms into the body immediately. In the original calculus we may have the term $u\{z_1\}\{z_2\}[z_1, z_2 \twoheadleftarrow \lambda c\langle t_1, t_2\rangle\overline{[\Gamma]}]$ where u is a term with variables z_1, z_2, but in our calculus we would instead have $u\{f_1\langle\,\vec{x_1}\,\rangle.t_1\}\{f_2\langle\,\vec{x_2}\,\rangle.t_2\}[f_1\langle\,\vec{x_1}\,\rangle, f_2\langle\,\vec{x_2}\,\rangle \,|\, c\langle\,c\,\rangle\,\overline{[\Gamma]}]$ where the distributor captures the phantom-variables f_1 and f_2, and x_1 and x_2 are the free variables of t_1 and t_2 that are captured by $\overline{[\Gamma]}$. The difference can also be illustrated in the proof theory. We introduce the distributor when an abstraction is duplicated as in Figure 3.5a. The introduced contraction inference rule would traverse up the derivation until it reached the abstraction inference rule that introduces the abstraction we are duplicating. At this point, we generate multiple independant derivations that correspond to the terms in the tuple. We use these derviations when eliminating the distributor and complete duplication as in Figure 3.5b. In our calculus, these derivations are pushed underneath the the distribution inference rule, and the idea is that we move the distributor up until it meets the abstraction as in Figure 3.5c. This becomes an important change when including the switch inference rule into the system, as it allows for more elegant rewrite steps. This will be explained in more detail in Section 3.3.3.

Below we show some examples of pre-terms and terms

- Pre-Terms (not Terms)
  - $c\langle\, x \,\rangle.y$.
    Violating condition 3 in Definition 5.
  - $x\, y[x, z \leftarrow w]$.
    Violating condition 4
  - $e_2\langle\, w_2 \,\rangle.w_2\,((e_1\langle\, w_1 \,\rangle.w_1)\, z)[e_1\langle\, w_1 \,\rangle, e_2\langle\, w_2 \,\rangle \,|\, c\langle\, z \,\rangle\, [w_1, w_2 \leftarrow x\langle\, x \,\rangle.x\, y]]$
    Violating condition $6(b)$
- Terms
  - $x\langle\, x \,\rangle.x$
  - $x\langle\, x \,\rangle.(y\langle\, y \,\rangle.x_1\,(x_2\, y)[x_1, x_2 \leftarrow x])$
  - $e_2\langle\, w_2 \,\rangle.w_2\,((e_1\langle\, w_1 \,\rangle.w_1)\, z)[e_1\langle\, w_1 \,\rangle, e_2\langle\, w_2 \,\rangle \,|\, c\langle\, c \,\rangle\, [w_1, w_2 \leftarrow x\langle\, x \,\rangle.x\, c]]$

Compilation and Readback

We now define the translations between Λ_a^S and the original λ-calculus. First we define the interpretation $\Lambda \to \Lambda_a^S$ (*compilation*). Intuitively, it replaces each abstraction $\lambda x.-$ with the term $x\langle\, x \,\rangle.-[x_1, \ldots, x_n \leftarrow x]$ where $x_1, \ldots, x_n$ replace the occurrences of x. Actual substitutions are denoted as $\{t/x\}$. Let $|\, M \,|_x$ denote the number of occurrences of x in M, and if $|\, M \,|_x = n$ let $M\frac{n}{x}$ denote M with the occurrences of x by fresh, distinct variables $x^1, \ldots, x^n$. First, the translation of a λ-term M is $(\!|\, M \,|\!)$, defined below

Definition 6 (Compilation)**.** *The interpretation of λ terms, $(\!|\, \Lambda \,|\!)' : \Lambda \to \Lambda_a^S$, is defined as*

$$(\!|\, M\frac{n_1}{x_1} \ldots \frac{n_k}{x_k} \,|\!)'[x_1^1, \ldots, x_1^{n_1} \leftarrow x_1] \ldots [x_k^1, \ldots, x_k^{n_k} \leftarrow x_k]$$

where $x_1, \ldots, x_k$ are the free variables of M such that $|\, M \,|_{x_i} = n_i > 1$ and $(\!|-|\!)'$ is defined on closed λ-terms as:

$$\begin{aligned}
(\!|\, x \,|\!)' &= x \\
(\!|\, M\, N \,|\!)' &= (\!|\, M \,|\!)'\, (\!|\, N \,|\!)' \\
(\!|\, \lambda x.M \,|\!)' &= \begin{cases} x\langle\, x \,\rangle.(\!|\, M \,|\!)' & \text{if } |\, M \,|_x = 1 \\ x\langle\, x \,\rangle.(\!|\, M\frac{n}{x} \,|\!)'[x^1, \ldots, x^n \leftarrow x] & \text{if } |\, M \,|_x = n \neq 1 \end{cases}
\end{aligned}$$

As an example we compile the SKI combinators introduced by Curry in [Cur30]

$$\begin{aligned}
(\!|\, \lambda x.x \,|\!) &= x\langle\, x \,\rangle.x \\
(\!|\, \lambda x.\lambda y.x \,|\!) &= x\langle\, x \,\rangle.y\langle\, y \,\rangle.(x[\leftarrow y]) \\
(\!|\, \lambda x \lambda y \lambda z x\, z\, (y\, z) \,|\!) &= x\langle\, x \,\rangle.y\langle\, y \,\rangle.z\langle\, z \,\rangle.x\, z_1\, (y\, z_2)[z_1, z_2 \leftarrow z]
\end{aligned}$$

To define the readback interpretation $[\![-]\!]$, we first discuss why it is not so simple. The complications rise from the distributor and phantom-abstractions. The distributor is duplicating an abstraction and captures phantom-variables of the phantom-abstractions (its

partial duplicates), and the environment in the distributor captures the variables listed in the cover of those phantom-abstractions. When translating this into the λ-calculus, we need to remember the (phantom-)variable bindings. When converting each phantom-abstraction into the original, it is not enough to convert them to the original abstraction variable name i.e. $[\![u[e_1\langle\, w_1 \,\rangle, e_2\langle\, w_2 \,\rangle \,|\, c\langle\, c \,\rangle\, [w_1, w_2 \leftarrow c]]]\!] = [\![u]\!]\{w_1/c\}\{w_2/c\}\{\lambda c/\lambda e_1\}\{\lambda c/\lambda e_2\}$ where $\{\lambda c/\lambda e\}$ renames the binding variable of an abstraction e.g. $(\lambda x.x)\{\lambda y/\lambda x\} = \lambda y.x$. Example 7 would then be converted as $\lambda c.\lambda c.c\,c$, thus the bindings of the variables are not maintained during translation.

Therefore, in the translation we need to keep track of the bindings so that in the case of interpreting a phantom-abstraction, we convert it into an abstraction and use the phantom-variable as the binding variable. To achieve this, we need to keep track of the bindings of phantom-variables. This will be the purpose of the map γ. As we traverse through the term we keep track of which abstraction a phantom-abstraction is a duplicate of. Then when it is time to evaluate the phantom-abstraction, we analyse the variables in the tuple. These variables will have terms substituted into them (as would happen during reduction). These terms being substituted in potentially have the variable we need to bind, and in such case we rename this variable to the name of the phantom-variable. This is done with the help of σ, which can be interpretated as an organised collection of substitutions. If we instead used regular subtitutions, we would need to wait for them to reach the phantom-variables before we continue as they might have the variable we need to bind.

Example 7. *Let $e_1\langle\, w_1 \,\rangle$ and $e_2\langle\, w_2 \,\rangle$ be captured by the same distributor.*

$$\begin{aligned} & (x\langle\, x \,\rangle.e_1\langle\, w_1 \,\rangle.x\, w_1)\, y\langle\, y \,\rangle.e_2\langle\, w_2 \,\rangle.y\, w_2 \\ \rightsquigarrow_\beta\ & e_1\langle\, w_1 \,\rangle.(y\langle\, y \,\rangle.e_2\langle\, w_2 \,\rangle.y\, w_2)\, w_1 \\ \rightsquigarrow_\beta\ & e_1\langle\, w_1 \,\rangle.e_2\langle\, w_2 \,\rangle.w_1\, w_2 \end{aligned}$$

We here discuss the notations used for the maps needed for the readback interpretation. We provide here the definition for the map from variables to Λ-terms $\sigma : V \to \Lambda$. The definitions for the map from variables to variables γ is defined in a similar way.

Definition 8. *We define a function $\sigma : V \to \Lambda$, which we denote as an infinite set of pairs of variables and Λ-terms. Given a variable x, $\sigma(x) = M$ such that $(x \mapsto M) \in \sigma$ and there does not exist N where $M \neq N$ and $(x \mapsto N) \in \sigma$.*

We use the following notation to make these functions more concise.

Notation 9. *We write $\sigma = \{x_1 \mapsto M_1, \ldots, x_n \mapsto M_n\}$ to mean the map where for $x_i \in \{x_1, \ldots, x_n\}$, $\sigma(x_i) = M_i$ and for $y \notin \{x_1, \ldots, x_n\}$, $\sigma(y) = y$.*

We also may need to combine maps to ontain a new map. This can only be done if, for all variables, the two different maps do not return two different terms where either of the terms is the variable itself.

Definition 10. *Given maps σ_1 and σ_2, such that for all variables x, either (*a*) $\sigma_1(x) = x = \sigma_2(x)$, (*b*) $\sigma_1(x) = M$ and $\sigma_2(x) = x$, or (*c*) $\sigma_1(x) = x$ and $\sigma_2(x) = M$ for some term M.*

Let A be the set of variables such that if $x \in A$ then $\sigma_1(x) \neq x$ and $\sigma_2(x) = x$, and let B be the set of variables for if $y \in B$ then $\sigma_1(y) = x$ and $\sigma_2(y) \neq y$.

The union of these maps $\sigma_1 \cup \sigma_2$ is defined as

$$\bigcup_{x \in A} \sigma_1(x) \cup \bigcup_{y \in B} \sigma_2(y)$$

Example 11. *Let $\sigma_1 = \{x_1 \mapsto N_1, x_2 \mapsto N_2\}$ and $\sigma_2 = \{y_1 \mapsto M_1\}$. Then $\sigma_1 \cup \sigma_2 = \{x_1 \mapsto N_1, x_2 \mapsto N_2, y_1 \mapsto M_1\}$*

Example 12. *Let $\sigma_1 = \{x_1 \mapsto N_1, x_2 \mapsto N_2\}$ and $\sigma_2 = \{x_1 \mapsto N_2\}$. Then $\sigma_1 \cup \sigma_2$ is undefined.*

When using the map σ as part of the translation, the intuition is that for all bound variables x in the term we are translatings, it should be that $\sigma(x) = x$. The map $\gamma : V \to V$ is defined similarly, and the purpose is to keep track of the binding of phantom-variables.

Definition 13. *The interpretation $[\![- \,|\, \frac{=}{=}]\!] : \Lambda_a^S \times (V \to \Lambda) \times (V \to V) \to \Lambda$ is defined as*

$$[\![x \,|\, \frac{\sigma}{\gamma}]\!] = \sigma(x)$$

$$[\![s\,t \,|\, \frac{\sigma}{\gamma}]\!] = [\![s \,|\, \frac{\sigma}{\gamma}]\!]\,[\![t \,|\, \frac{\sigma}{\gamma}]\!]$$

$$[\![c\langle\, c \,\rangle.t \,|\, \frac{\sigma}{\gamma}]\!] = \lambda c.[\![t \,|\, \frac{\sigma'}{\gamma}]\!]$$
$$\text{where } \sigma' = \sigma - \{c \mapsto M\}$$

$$[\![c\langle\, x_1, \ldots, x_n \,\rangle.t \,|\, \frac{\sigma}{\gamma}]\!] = \lambda c.[\![t \,|\, \frac{\sigma'}{\gamma}]\!]$$
$$\text{let } \sigma_1 = \sigma - \{x_1, ..., x_n\}$$
$$\text{then } \sigma' = \sigma_1 \cup \{x_1 \mapsto \sigma(x_1)\{c/d\}, \cup ... \cup, x_n \mapsto \sigma(x_n)\{c/d\}\}$$
$$\text{where } d = \gamma(c)$$

$$[\![u[x_1, \ldots, x_n \leftarrow t] \,|\, \frac{\sigma}{\gamma}]\!] = [\![u \,|\, \frac{\sigma'}{\gamma}]\!]$$
$$\text{let } \sigma_1 = \sigma - \{x_1, \ldots, x_n\}$$
$$\text{where } \sigma' = \sigma_1 \cup \{x_i \mapsto [\![t \,|\, \frac{\sigma}{\gamma}]\!]\}_{1 \le i \le n}$$

$$[\![u[e_1\langle\, \vec{w}_1 \,\rangle, \ldots, e_n\langle\, \vec{w}_n \,\rangle \,|\, c\langle\, c \,\rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma}{\gamma}]\!] = [\![u\overline{[\Gamma]} \,|\, \frac{\sigma}{\gamma'}]\!]$$
$$\text{where } \gamma' = \gamma \cup \{e_1 \mapsto c, \ldots, e_n \mapsto c\}$$

$$[\![u[e_1\langle\, \vec{w}_1 \,\rangle, \ldots, e_n\langle\, \vec{w}_n \,\rangle \,|\, c\langle\, x_1, \ldots, x_m \,\rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma}{\gamma}]\!] = [\![u\overline{[\Gamma]} \,|\, \frac{\sigma'}{\gamma'}]\!]$$
$$\text{where } \gamma' = \gamma \cup \{e_1 \mapsto c, \ldots, e_n \mapsto c\}$$
$$\sigma_1 = \sigma - \{x_1, \ldots, x_m\}$$
$$\sigma' = \sigma_1 \cup \{x_1 \mapsto M_1\{c/\gamma(c)\}, \ldots, x_m \mapsto M_m\{c/\gamma(c)\}\}$$

Definition 14. *We say* $[\![t]\!] = [\![t \mid \frac{I}{I}]\!]$ *where* I *is the identity function.*

Notions of Equivalence

In the term calculus, we consider terms equal up to the congruence induced by the exchange of closures. Consider the term $t[\Gamma_1][\Gamma_2]$ where $[\Gamma_1]$ and $[\Gamma_2]$ are both closures. Then $t[\Gamma_1][\Gamma_2] \sim t[\Gamma_2][\Gamma_1]$ iff $[\Gamma_2]$ only binds variables and phantom-variables located in t. This equivalence is essential to the rewriting theory.

We also consider terms equal up to symmetry of contraction. We consider the sequence of variables xs modulo permutations. Let $\vec{x}$ be a list of variables and let $\vec{x_P}$ be a permutation of that list, then the following terms are considered equal.

$$u[\vec{x} \leftarrow t] \sim u[\vec{x_P} \leftarrow t]$$
$$c\langle\, \vec{x} \,\rangle.t \sim c\langle\, \vec{x_P} \,\rangle.t$$

3.1.2 Operations

The spinal atomic λ-calculus is a refinement of the original [GHP13], where during reduction, duplication of subterms proceeds on individual constructors, similar to Lamping's optimal graph reduction [Lam90]. The reduction rules (defined later in Section 3.2) will make use of some operations. We will here define the three different operations, *substitution*, *book-keeping*, and *exorcism*.

The purpose of the operation of substitution, as expected, is to replace free variables x occurring in a term t with a subterm u, written as $t\{s/x\}$. Substitution in our calculus also changes the covers of phantom-abstractions if the variable we are substituting for is listed. For example $(c\langle\, x \,\rangle.t)\{s/x\}$, after the substitution moves under the phantom-abstraction the variable in the cover is replaced with all the free variables in s, i.e. $c\langle\, \vec{z} \,\rangle.t\{s/x\}$ where $\{\vec{z}\} = (s)_{fv}$.

In the case of the distributor, substitutions need to travel through the environment of the distributor and afterwards leave the distributor and travel through the term. This is because variables occurring in the term may be bound by a sharing in the distributor. When we define our reductions in Section 3.2, some of these reductions will generate substitutions, and some of these may occur within the environment of the distributor, therefore the substitutions need some way of 'escaping' the distributor.

We use $\overrightarrow{e\langle\, \vec{w} \,\rangle}$ to denote $e_1\langle\, w_1^1, \ldots, w_{k_1}^1 \,\rangle \ldots e_n\langle\, w_1^n, \ldots, w_{k_n}^n \,\rangle$ for some arbitrary n and k. We write $e\{e_i\langle\, \vec{w}_i \,\rangle\}$ to denote a list of covers $\overrightarrow{e\langle\, \vec{w} \,\rangle}$ where $e_i\langle\, \vec{w}_i \,\rangle$ occurs in the list.

Definition 15 (Substitution)**.** *The operation* substitution *is defined as*

$$\begin{aligned}
x\{s/x\} &= s \\
y\{s/x\} &= y \\
(u\,t)\{s/x\} &= (u\{s/x\})\,t\{s/x\} \\
(c\langle\, \vec{y} \,\rangle.t)\{s/x\} &= c\langle\, \vec{y} \,\rangle.t\{s/x\} \\
(c\langle\, \vec{y} \cdot x \,\rangle.t)\{s/x\} &= c\langle\, \vec{y} \cdot \vec{z} \,\rangle.t\{s/x\} \\
u[\vec{y} \leftarrow t]\{s/x\} &= u\{s/x\}[\vec{y} \leftarrow t\{s/x\}]
\end{aligned}$$

$$u[\overrightarrow{e\langle\,\vec{w}\,\rangle}\,|\,c\langle\,\vec{y}\,\rangle\,\overline{[\Gamma]}]\{s/x\} = u[\overrightarrow{e\langle\,\vec{w}\,\rangle}\,|\,c\langle\,\vec{y}\,\rangle\,\overline{[\Gamma]}\{s/x\}]$$
$$u[\overrightarrow{e\langle\,\vec{w}\,\rangle}\,|\,c\langle\,\vec{y}\cdot x\,\rangle\,\overline{[\Gamma]}]\{s/x\} = u[\overrightarrow{e\langle\,\vec{w}\,\rangle}\,|\,c\langle\,\vec{y}\cdot\vec{z}\,\rangle\,\overline{[\Gamma]}\{s/x\}]$$
$$u[\overrightarrow{e\langle\,\vec{w}\,\rangle}\,|\,c\langle\,\vec{y}\,\rangle\,\{s/x\}\overline{[\Gamma]}] = u\{s/x\}[\overrightarrow{e\langle\,\vec{w}\,\rangle}\,|\,c\langle\,\vec{y}\,\rangle\,\overline{[\Gamma]}]$$
$$u[e\{e_i\langle\,\vec{w}\cdot x\,\rangle\}\,|\,c\langle\,\vec{y}\,\rangle\,\{s/x\}\overline{[\Gamma]}] = u\{s/x\}[e\{e_i\langle\,\vec{w}\cdot\vec{z}\,\rangle\}\,|\,c\langle\,\vec{y}\,\rangle\,\overline{[\Gamma]}]$$

Where $\vec{z} = (s)_{fv}$

Although substitution performs some book-keeping on phantom-abstractions, we define an explicit notion that updates the variables stored in a free-cover i.e. for a term t, $e\langle\,\vec{x}\,\rangle \in (t)_{fc}$ then $e\langle\,\vec{y}\,\rangle \in (t\{\vec{y}/e\}_b)_{fc}$.

Definition 16 (Book-Keeping)**.** *The operation* book-keeping *is defined as*

$$x\{\vec{w}/e\}_b = x$$
$$s\,t\{\vec{w}/e\}_b = (s\{\vec{w}/e\}_b)\,t\{\vec{w}/e\}_b$$
$$e\langle\,\vec{z}\,\rangle.t\{\vec{w}/e\}_b = e\langle\,\vec{w}\,\rangle.t$$
$$(c\langle\,\vec{z}\,\rangle.t)\{\vec{w}/e\}_b = c\langle\,\vec{z}\,\rangle.t\{\vec{w}/e\}_b$$
$$u[\vec{z}\leftarrow t]\{\vec{w}/e\}_b = u\{\vec{w}/e\}_b[\vec{z}\leftarrow t\{\vec{w}/e\}_b]$$
$$u[\overrightarrow{f\langle\,\vec{y}\,\rangle}\,|\,e\langle\,\vec{z}\,\rangle\,\overline{[\Gamma]}]\{\vec{w}/e\}_b = u[\overrightarrow{f\langle\,\vec{y}\,\rangle}\,|\,e\langle\,\vec{w}\,\rangle\,\overline{[\Gamma]}]$$
$$u[\overrightarrow{f\langle\,\vec{y}\,\rangle}\,|\,c\langle\,\vec{z}\,\rangle\,\overline{[\Gamma]}]\{\vec{w}/e\}_b = u[\overrightarrow{f\langle\,\vec{y}\,\rangle}\,|\,c\langle\,\vec{z}\,\rangle\,\overline{[\Gamma]}\{\vec{w}/e\}_b]$$
$$u[\overrightarrow{f\langle\,\vec{y}\,\rangle}\,|\,c\langle\,\vec{z}\,\rangle\,\{\vec{w}/e\}_b\overline{[\Gamma]}] = u\{\vec{w}/e\}_b[\overrightarrow{f\langle\,\vec{y}\,\rangle}\,|\,c\langle\,\vec{z}\,\rangle\,\overline{[\Gamma]}]$$

The last operation we introduce is called *exorcism* $\{c\langle\,\vec{x}\,\rangle\}_e$. We perform exorcisms on phantom-abstractions to convert them to abstractions. Intuitively, this will be performed on phantom-abstractions with phantom-variables bound to a distributor when said distributor is eliminated. It converts phantom-abstractions to abstractions by introducing a sharing of the phantom-variable that captures the variables in the cover, i.e. $c\langle\,\vec{x}\,\rangle.t\{c\langle\,\vec{x}\,\rangle\}_e = c\langle\,c\,\rangle.t[\vec{x}\leftarrow c]$.

Definition 17 (Exorcism)**.** *The operation* exorcism *is defined as*

$$y\{c\langle\,\vec{x}\,\rangle\}_e = y$$
$$s\,t\{c\langle\,\vec{x}\,\rangle\}_e = (s\{c\langle\,\vec{x}\,\rangle\}_e)\,t\{c\langle\,\vec{x}\,\rangle\}_e$$
$$c\langle\,\vec{x}\,\rangle.t\{c\langle\,\vec{x}\,\rangle\}_e = c\langle\,c\,\rangle.t[\vec{x}\leftarrow c]$$
$$d\langle\,\vec{y}\,\rangle.t\{c\langle\,\vec{x}\,\rangle\}_e = d\langle\,\vec{y}\,\rangle.t\{c\langle\,\vec{x}\,\rangle\}_e$$
$$u[\vec{y}\leftarrow t]\{c\langle\,\vec{x}\,\rangle\}_e = u\{c\langle\,\vec{x}\,\rangle\}_e[\vec{y}\leftarrow t\{c\langle\,\vec{x}\,\rangle\}_e]$$
$$u[\overrightarrow{e\langle\,\vec{w}\,\rangle}\,|\,c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}]\{c\langle\,\vec{x}\,\rangle\}_e = u[\overrightarrow{e\langle\,\vec{w}\,\rangle}\,|\,c\langle\,c\,\rangle\,\overline{[\Gamma]}[\vec{x}\leftarrow c]]$$
$$u[\overrightarrow{e\langle\,\vec{w}\,\rangle}\,|\,d\langle\,\vec{y}\,\rangle\,\overline{[\Gamma]}]\{c\langle\,\vec{x}\,\rangle\}_e = u[\overrightarrow{e\langle\,\vec{w}\,\rangle}\,|\,d\langle\,\vec{y}\,\rangle\,\overline{[\Gamma]}\{c\langle\,\vec{x}\,\rangle\}_e]$$
$$u[\overrightarrow{e\langle\,\vec{w}\,\rangle}\,|\,d\langle\,\vec{y}\,\rangle\,\{c\langle\,\vec{x}\,\rangle\}_e\overline{[\Gamma]}] = u\{c\langle\,\vec{w}\,\rangle\}_e[\overrightarrow{e\langle\,\vec{w}\,\rangle}\,|\,d\langle\,\vec{y}\,\rangle\,\overline{[\Gamma]}]$$

Proposition 18. *Given $M \in \Lambda$ such that for all $v \in V$, $\gamma(v) \notin (M)_{fv}$ and $\sigma(x) = x$, the translation $[\![- \,|\, \frac{\sigma}{\gamma}]\!]$ commutes with substitution (Def 15) in the following way*

$$[\![u \,|\, \frac{\sigma'}{\gamma}]\!] = [\![u \,|\, \frac{\sigma}{\gamma}]\!]\{M/x\}$$

where $\sigma' = \sigma\{M/x\} \cup \{x \mapsto M\}$

Proof. We prove this by induction on u

Base Case: Variable
$[\![x \,|\, \frac{\sigma}{\gamma}]\!]\{M/x\} = x\{M/x\} = M = [\![x \,|\, \frac{\sigma'}{\gamma}]\!]$

$[\![y \,|\, \frac{\sigma}{\gamma}]\!]\{M/x\} = N\{M/x\} = [\![y \,|\, \frac{\sigma'}{\gamma}]\!]$

Inductive Case: Application
$[\![s\,t \,|\, \frac{\sigma}{\gamma}]\!]\{M/x\} = [\![s \,|\, \frac{\sigma}{\gamma}]\!]\{M/x\}\,[\![t \,|\, \frac{\sigma}{\gamma}]\!]\{M/x\} \overset{\text{I.H.}}{=} [\![s \,|\, \frac{\sigma}{\gamma}]\!]\,[\![t \,|\, \frac{\sigma'}{\gamma}]\!] = [\![s\,t \,|\, \frac{\sigma'}{\gamma}]\!]$

Inductive Case: Abstraction
$[\![c\langle\, c \,\rangle.t \,|\, \frac{\sigma}{\gamma}]\!]\{M/x\} = \lambda c.[\![t \,|\, \frac{\sigma}{\gamma}]\!]\{M/x\} \overset{\text{I.H.}}{=} \lambda c.[\![t \,|\, \frac{\sigma'}{\gamma}]\!] = [\![c\langle\, c \,\rangle.t \,|\, \frac{\sigma'}{\gamma}]\!]$

Inductive Case: Phantom-Abstraction
$[\![c\langle\, x_1, \ldots, x_n \,\rangle.t \,|\, \frac{\sigma}{\gamma}]\!]\{M/x\} = (\lambda c.[\![t \,|\, \frac{\sigma''}{\gamma}]\!])\{M/x\} = \lambda c.[\![t \,|\, \frac{\sigma''}{\gamma}]\!]\{M/x\} \overset{\text{I.H.}}{=} \lambda c.[\![t \,|\, \frac{\sigma'''}{\gamma}]\!]$
$= [\![c\langle\, x_1, \ldots, x_n \,\rangle.t \,|\, \frac{\sigma'}{\gamma}]\!]$
where
$\sigma = \sigma_1 \cup \{x_1 \mapsto N_1, \ldots, x_n \mapsto N_n\}$
$\sigma'' = \sigma_1 \cup \{x_1 \mapsto N_1\{c/d\}, \ldots, x_n \mapsto N_n\{c/d\}\}$
$\sigma''' = \sigma''\{M/x\} \cup \{x \mapsto M\}$
$\sigma''' = \sigma_1\{M/x\} \cup \{x_1 \mapsto N_1\{M/x\}\{c/d\}, \ldots, x_n \mapsto N_n\{M/x\}\{c/d\}, x \mapsto M\}$

Inductive Case: Sharing
$[\![u[z_1, \ldots, z_n \leftarrow t] \,|\, \frac{\sigma}{\gamma}]\!]\{M/x\} = [\![u \,|\, \frac{\sigma''}{\gamma}]\!]\{M/x\} \overset{\text{I.H.}}{=} [\![u \,|\, \frac{\sigma'''}{\gamma}]\!] = [\![u[z_1, \ldots, z_n \leftarrow t] \,|\, \frac{\sigma'}{\gamma}]\!]$
where
$\sigma'' = \sigma \cup \bigcup_{1 \le i \le n} \{z_i \mapsto [\![t \,|\, \frac{\sigma}{\gamma}]\!]\}$
$\sigma''' = \sigma\{M/x\} \cup \bigcup_{1 \le i \le n} \{z_i \mapsto [\![t \,|\, \frac{\sigma\{x/M\} \cup \{x \mapsto M\}}{\gamma}]\!], x \mapsto M\}$

Inductive Case: Distributor 1
$[\![u[e_1\langle\, \vec{w}_1 \,\rangle, \ldots, e_n\langle\, \vec{w}_n \,\rangle \,|\, c\langle\, c \,\rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma}{\gamma}]\!]\{M/x\}$
$= [\![u\overline{[\Gamma]} \,|\, \frac{\sigma}{\gamma'}]\!]\{M/x\} \overset{\text{I.H.}}{=} [\![u\overline{[\Gamma]} \,|\, \frac{\sigma'}{\gamma'}]\!]$
$= [\![u[e_1\langle\, \vec{w}_1 \,\rangle, \ldots, e_n\langle\, \vec{w}_n \,\rangle \,|\, c\langle\, c \,\rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma'}{\gamma}]\!]$
where
$\gamma' = \gamma \cup \{e_1 \mapsto c, \ldots, e_n \mapsto c\}$

Inductive Case: Distributor 2
$[\![u[e_1\langle\, \vec{w}_1 \,\rangle, \ldots, e_n\langle\, \vec{w}_n \,\rangle \,|\, c\langle\, \vec{x} \,\rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma}{\gamma}]\!]\{M/x\}$

$= [\![u\overline{[\Gamma]} \,|\, \frac{\sigma''}{\gamma'}]\!]\{M/x\} \overset{\text{I.H.}}{=} [\![u\overline{[\Gamma]} \,|\, \frac{\sigma'''}{\gamma'}]\!]$
$= [\![u[e_1\langle \vec{w}_1 \rangle, \ldots, e_n\langle \vec{w}_n \rangle \,|\, c\langle \vec{x} \rangle \overline{[\Gamma]}] \,|\, \frac{\sigma'}{\gamma}]\!]$
where
$\gamma' = \gamma \cup \{e_1 \mapsto c, \ldots, e_n \mapsto c\}$ □

Proposition 19. *Substitution commutes with the translation in the following way*

$$[\![u\{t/x\} \,|\, \frac{\sigma}{\gamma}]\!] = [\![u \,|\, \frac{\sigma'}{\gamma}]\!]$$

where $\sigma' = \sigma \cup \{x \mapsto [\![t \,|\, \frac{\sigma}{\gamma}]\!]\}$

Proof. We prove this by induction on u

Base Case: Variable
$[\![x\{t/x\} \,|\, \frac{\sigma}{\gamma}]\!] = [\![t \,|\, \frac{\sigma}{\gamma}]\!] = [\![x \,|\, \frac{\sigma'}{\gamma}]\!]$

$[\![y \,|\, \frac{\sigma}{\gamma}]\!] = \sigma(y) = \sigma'(y) = [\![y \,|\, \frac{\sigma'}{\gamma}]\!]$

Inductive Case: Application
$[\![u\, s\{t/x\} \,|\, \frac{\sigma}{\gamma}]\!] = [\![u\{t/x\} \,|\, \frac{\sigma}{\gamma}]\!] [\![s\{t/x\} \,|\, \frac{\sigma}{\gamma}]\!] \overset{\text{I.H.}}{=} [\![u \,|\, \frac{\sigma'}{\gamma}]\!] [\![s \,|\, \frac{\sigma'}{\gamma}]\!] = [\![u\, s \,|\, \frac{\sigma'}{\gamma}]\!]$

Induvtive Case: Abstraction
$[\![(c\langle c \rangle.s)\{t/x\} \,|\, \frac{\sigma}{\gamma}]\!] = \lambda c.[\![s\{t/x\} \,|\, \frac{\sigma}{\gamma}]\!] \overset{\text{I.H.}}{=} \lambda c.[\![s \,|\, \frac{\sigma'}{\gamma}]\!] = [\![c\langle c \rangle.s \,|\, \frac{\sigma'}{\gamma}]\!]$

Inductive Case: Phantom-Abstraction
$[\![(c\langle x_1, \ldots, x_n \rangle.s)\{t/x\} \,|\, \frac{\sigma}{\gamma}]\!]$
Case: $x \in \{x_1, \ldots, x_n\}$
$= [\![(c\langle x_1, \ldots, x_n, x \rangle.s)\{t/x\} \,|\, \frac{\sigma}{\gamma}]\!] = [\![c\langle x_1, \ldots, x_n, y_1, \ldots, y_m \rangle.s\{t/x\} \,|\, \frac{\sigma}{\gamma}]\!]$
where $\{y_1, \ldots, y_m\} = (t)_{fv}$
$= \lambda c.[\![s\{t/x\} \,|\, \frac{\sigma''}{\gamma}]\!] \overset{\text{I.H.}}{=} \lambda c.[\![s \,|\, \frac{\sigma_1'''}{\gamma}]\!] \overset{\text{prop 18}}{=} \lambda c.[\![s \,|\, \frac{\sigma_2'''}{\gamma}]\!] = [\![c\langle x_1, \ldots, x_n, x \rangle.s \,|\, \frac{\sigma'}{\gamma}]\!]$
where $\sigma = \sigma'''' \cup \{x_1 \mapsto M_1, \ldots, y_m \mapsto N_m\}$
$\sigma'' = \sigma'''' \cup \{x_1 \mapsto M_1\{c/d\}, \ldots, y_m \mapsto N_m\{c/d\}\}$
where $d = \gamma(c)$
$\sigma_1''' = \sigma'' \cup \{x \mapsto [\![t \,|\, \frac{\sigma''}{\gamma}]\!]\}$
$\sigma_2''' = \sigma'''' \cup \{y_1 \mapsto N_1, \ldots, y_m \mapsto N_m, x_1 \mapsto M_1\{c/d\}, \ldots, x_n \mapsto M_n\{c/d\}, x \mapsto [\![t \,|\, \frac{\sigma'}{\gamma}]\!]\{c/d\}\}$

Case: $x \in \{x_1, \ldots, x_n\}$
$= [\![c\langle x_1, \ldots, x_n \rangle.s\{t/x\} \,|\, \frac{\sigma}{\gamma}]\!] = \lambda c.[\![s\{t/x\} \,|\, \frac{\sigma''}{\gamma}]\!] \overset{\text{I.H.}}{=} \lambda c.[\![t \,|\, \frac{\sigma'''}{\gamma}]\!] = [\![c\langle x_1, \ldots, x_n \rangle.s \,|\, \frac{\sigma'}{\gamma}]\!]$
where
$\sigma = \sigma'''' \cup \{x_1 \mapsto N_1, \ldots, x_n \mapsto N_n\}$
$\sigma'' = \sigma'''' \cup \{x_1 \mapsto N_1\{c/d\}, \ldots, x_n \mapsto N_n\{c/d\}\}$
$d = \gamma(c)$
$\sigma''' = \sigma'' \cup \{x \mapsto [\![t \,|\, \frac{\sigma''}{\gamma}]\!]\}$

Inductive Case: Sharing
$[\![u[z_1,\ldots,z_n \leftarrow s]\{t/x\} \,|\, \frac{\sigma}{\gamma}]\!] = [\![u\{t/x\}[z_1,\ldots,z_n \leftarrow s\{t/x\}] \,|\, \frac{\sigma}{\gamma}]\!] = [\![u\{t/x\} \,|\, \frac{\sigma''}{\gamma}]\!]$
$\overset{\text{I.H.}}{=} [\![u \,|\, \frac{\sigma'''}{\gamma}]\!] = [\![u[z_1,\ldots,z_n \leftarrow s] \,|\, \frac{\sigma'}{\gamma}]\!]$
where
$\sigma'' = \sigma \cup \{z_1 \mapsto [\![s\{t/x\} \,|\, \frac{\sigma}{\gamma}]\!], \ldots, z_n \mapsto [\![s\{t/x\} \,|\, \frac{\sigma}{\gamma}]\!]\}$
$\sigma''' = \sigma' \cup \{z_1 \mapsto [\![s \,|\, \frac{\sigma'}{\gamma}]\!], \ldots, z_n \mapsto [\![s \,|\, \frac{\sigma'}{\gamma}]\!]\}$

Inductive Case: Distributor 1
$[\![u[e_1\langle \vec{w}_1 \rangle, \ldots, e_n\langle \vec{w}_n \rangle \,|\, c\langle c \rangle \overline{[\Gamma]}]\{t/x\} \,|\, \frac{\sigma}{\gamma}]\!]$
$= [\![u\overline{[\Gamma]}\{t/x\} \,|\, \frac{\sigma}{\gamma'}]\!] \overset{\text{I.H.}}{=} [\![u\overline{[\Gamma]} \,|\, \frac{\sigma'}{\gamma'}]\!]$
$= [\![u[e_1\langle \vec{w}_1 \rangle, \ldots, e_n\langle \vec{w}_n \rangle \,|\, c\langle \vec{x} \rangle \overline{[\Gamma]}] \,|\, \frac{\sigma'}{\gamma}]\!]$
where
$\gamma' = \gamma \cup \{e_1 \mapsto c, \ldots, e_n \mapsto c\}$

Inductive Case: Distributor 2
$[\![u[e_1\langle \vec{w}_1 \rangle, \ldots, e_n\langle \vec{w}_n \rangle \,|\, c\langle \vec{x} \rangle \overline{[\Gamma]}]\{t/x\} \,|\, \frac{\sigma}{\gamma}]\!]$
$= [\![u\overline{[\Gamma]}\{t/x\} \,|\, \frac{\sigma''}{\gamma'}]\!] \overset{\text{I.H.}}{=} [\![u\overline{[\Gamma]} \,|\, \frac{\sigma'''}{\gamma'}]\!]$
$= [\![u[e_1\langle \vec{w}_1 \rangle, \ldots, e_n\langle \vec{w}_n \rangle \,|\, c\langle \vec{x} \rangle \overline{[\Gamma]}] \,|\, \frac{\sigma'}{\gamma}]\!]$
where
$\gamma' = \gamma \cup \{e_1 \mapsto c, \ldots, e_n \mapsto c\}$ □

Proposition 20. *Book-keeping commutes with the translation in the following way*
if $c\langle y_1,\ldots,y_m \rangle. \in (u)_{fc}$ *such that* $\{x_1,\ldots,x_n\} \subset \{y_1,\ldots,y_m\}$
and for those $z \in \{y_1,\ldots,y_m\}/\{x_1,\ldots,x_n\}$, $\gamma(c) \notin (\sigma(z))_{fv}$
or if simply $\{x_1,\ldots,x_n\} \cap (u)_{fv} = \{\}$

$$[\![u\{x_1,\ldots,x_n/c\}_b \,|\, \frac{\sigma}{\gamma}]\!] = [\![u \,|\, \frac{\sigma}{\gamma}]\!]$$

Proof. We prove this by induction on u

Base Case: Variable
$[\![x\{x_1,\ldots,x_n/c\}_b \,|\, \frac{\sigma}{\gamma}]\!] = [\![x \,|\, \frac{\sigma}{\gamma}]\!] = \sigma(x) = \sigma'(x) = [\![x \,|\, \frac{\sigma'}{\gamma'}]\!]$
Since is cannot be that $x \in \{x_1,\ldots,x_n\}$

Base Case: Phantom-Abstraction
$[\![(c\langle y_1,\ldots,y_m \rangle.t)\{x_1,\ldots,x_n/c\}_b \,|\, \frac{\sigma}{\gamma}]\!] = [\![c\langle x_1,\ldots,x_n \rangle.t \,|\, \frac{\sigma}{\gamma}]\!]$
$= \lambda c.[\![t \,|\, \frac{\sigma''}{\gamma}]\!] = \lambda c.[\![t \,|\, \frac{\sigma''}{\gamma'}]\!] = [\![c\langle y_1,\ldots,y_m \rangle.t \,|\, \frac{\sigma'}{\gamma'}]\!]$
where
$\sigma = \sigma_1 \cup \{x_1 \mapsto M_1, \ldots, x_n \mapsto M_n\}$
$\sigma'' = \sigma_1 \cup \{x_1 \mapsto M_1\{c/d\}, \ldots, x_n \mapsto M_n\{c/d\}\}$
$\gamma(c) = d$
Note: due to condition of Proposition any $\{y_i \mapsto M_i\{c/d\}\} = \{y_i \mapsto M_i\}$

Base Case: Distributor
$[\![u[e_1\langle \vec{w}_1 \rangle, \ldots, e_n\langle \vec{w}_n \rangle \,|\, c\langle y_1, \ldots, y_m \rangle \overline{[\Gamma]}]\{x_1, \ldots, x_n/c\}_b \,|\, \frac{\sigma}{\gamma}]\!]$
$= [\![u[e_1\langle \vec{w}_1 \rangle, \ldots, e_n\langle \vec{w}_n \rangle \,|\, c\langle x_1, \ldots, x_n \rangle \overline{[\Gamma]}] \,|\, \frac{\sigma}{\gamma}]\!] = [\![u\overline{[\Gamma]} \,|\, \frac{\sigma'}{\gamma'}]\!]$
$= [\![u[e_1\langle \vec{w}_1 \rangle, \ldots, e_n\langle \vec{w}_n \rangle \,|\, c\langle y_1, \ldots, y_m \rangle \overline{[\Gamma]}] \,|\, \frac{\sigma}{\gamma}]\!]$
where $\gamma' = \gamma \cup \{e_1 \mapsto c, \ldots, e_n \mapsto c\}$
$\sigma = \sigma_1 \cup \{x_1 \mapsto M_1, \ldots, x_n \mapsto M_n\}$
$\sigma' = \sigma_1 \cup \{x_1 \mapsto M_1\{c/\gamma(c)\}, \ldots, x_n \mapsto M_n\{c/\gamma(c)\}\}$

Inductive Case: Application
$[\![(s\,t)\{x_1, \ldots, x_n/c\}_b \,|\, \frac{\sigma}{\gamma}]\!] = [\![s\{x_1, \ldots, x_n/c\}_b \,|\, \frac{\sigma}{\gamma}]\!]\,[\![t\{x_1, \ldots, x_n/c\}_b \,|\, \frac{\sigma}{\gamma}]\!]$
$\overset{\text{I.H.}}{=} [\![s \,|\, \frac{\sigma}{\gamma}]\!]\,[\![t \,|\, \frac{\sigma}{\gamma}]\!] = [\![s\,t \,|\, \frac{\sigma}{\gamma}]\!]$

Inductive Case: Abstraction
$[\![(z\langle z \rangle.t)\{x_1, \ldots, x_n/c\}_b \,|\, \frac{\sigma}{\gamma}]\!] = \lambda z.[\![t\{x_1, \ldots, x_n/c\}_b \,|\, \frac{\sigma}{\gamma}]\!] \overset{\text{I.H.}}{=} \lambda z.[\![t \,|\, \frac{\sigma}{\gamma}]\!] = [\![z\langle z \rangle.t \,|\, \frac{\sigma}{\gamma}]\!]$

Inductive Case: Phantom-Abstraction
$[\![(d\langle z_1, \ldots, z_m \rangle.t)\{x_1, \ldots, x_n/c\}_b \,|\, \frac{\sigma}{\gamma}]\!] = \lambda d.[\![t\{x_1, \ldots, x_n/c\}_b \,|\, \frac{\sigma'}{\gamma}]\!]$
$\overset{\text{I.H.}}{=} \lambda d.[\![t \,|\, \frac{\sigma'}{\gamma}]\!] = [\![d\langle z_1, \ldots, z_m \rangle.t \,|\, \frac{\sigma}{\gamma}]\!]$

Inductive Case: Sharing
$[\![u[z_1, \ldots, z_m \leftarrow t]\{x_1, \ldots, x_n/c\}_b \,|\, \frac{\sigma}{\gamma}]\!]$
$= [\![u\{x_1, \ldots, x_n/c\}_b[z_1, \ldots, z_m \leftarrow t\{x_1, \ldots, x_n/c\}_b] \,|\, \frac{\sigma}{\gamma}]\!]$
$= [\![u\{x_1, \ldots, x_n/c\}_b \,|\, \frac{\sigma'}{\gamma}]\!] \overset{\text{I.H.}}{=} [\![u \,|\, \frac{\sigma''}{\gamma}]\!] = [\![u[z_1, \ldots, z_m \leftarrow t] \,|\, \frac{\sigma}{\gamma}]\!]$

Inductive Case: Distributor
$[\![u[e_1\langle \vec{w}_1 \rangle, \ldots, e_m\langle \vec{w_m} \rangle \,|\, d\langle d \rangle \overline{[\Gamma]}]\{x_1, \ldots, x_n/c\}_b \,|\, \frac{\sigma}{\gamma}]\!]$
$= [\![u[e_1\langle \vec{w}_1 \rangle, \ldots, e_m\langle \vec{w_m} \rangle \,|\, d\langle d \rangle \overline{[\Gamma]}\{x_1, \ldots, x_n/c\}_b] \,|\, \frac{\sigma}{\gamma}]\!]$
$= [\![u\overline{[\Gamma]}\{x_1, \ldots, x_n/c\}_b \,|\, \frac{\sigma}{\gamma'}]\!] \overset{\text{I.H.}}{=} [\![u\overline{[\Gamma]} \,|\, \frac{\sigma}{\gamma'}]\!]$
$= [\![u[e_1\langle \vec{w}_1 \rangle, \ldots, e_m\langle \vec{w_m} \rangle \,|\, d\langle d \rangle \overline{[\Gamma]}] \,|\, \frac{\sigma}{\gamma}]\!]$ □

Proposition 21. *Exorcisms commute with the translation in the following way if* $c\langle x_1, \ldots, x_n \rangle. \in (u)_{fc}$ *or* $\{x_1, \ldots, x_n\} \cap (u)_{fv} = \{\}$

$$[\![u\{c\langle x_1, \ldots, x_n \rangle\}_e \,|\, \frac{\sigma}{\gamma}]\!] = [\![u \,|\, \frac{\sigma'}{\gamma}]\!]$$

where
$\sigma' = \sigma \cup (x_1 \mapsto c) \cup \cdots \cup (x_n \mapsto c)$

Proof. We prove this by induction on u

Base Case: Variable
$[\![z\{c\langle x_1, \ldots, x_n \rangle\}_e \,|\, \frac{\sigma}{\gamma}]\!] = [\![z \,|\, \frac{\sigma}{\gamma}]\!] = \sigma(z) = \sigma'(z) = [\![z \,|\, \frac{\sigma'}{\gamma}]\!]$

Base Case: Phantom-Abstraction
$[\![(c\langle x_1,\ldots,x_n\rangle.t)\{c\langle x_1,\ldots,x_n\rangle\}_e \,|\, \frac{\sigma}{\gamma}]\!] = [\![c\langle c\rangle.t[x_1,\ldots,x_n \leftarrow c] \,|\, \frac{\sigma}{\gamma}]\!]$
$= \lambda c.[\![t[x_1,\ldots,x_n \leftarrow c] \,|\, \frac{\sigma}{\gamma}]\!] = \lambda c.[\![t \,|\, \frac{\sigma'}{\gamma}]\!] = [\![c\langle x_1,\ldots,x_n\rangle.t \,|\, \frac{\sigma'}{\gamma}]\!]$

Base Case: Distributor
$[\![u[e_1\langle \vec{w_1}\rangle,\ldots,e_m\langle \vec{w_m}\rangle \,|\, c\langle x_1,\ldots,x_n\rangle \overline{[\Gamma]}]\{c\langle x_1,\ldots,x_n\rangle\}_e \,|\, \frac{\sigma}{\gamma}]\!]$
$= [\![u[e_1\langle \vec{w_1}\rangle,\ldots,e_m\langle \vec{w_m}\rangle \,|\, c\langle c\rangle \overline{[\Gamma]}[x_1,\ldots,x_n \leftarrow c]] \,|\, \frac{\sigma}{\gamma}]\!]$
$= [\![u\overline{[\Gamma]}[x_1,\ldots,x_n \leftarrow c] \,|\, \frac{\sigma}{\gamma'}]\!] = [\![u\overline{[\Gamma]} \,|\, \frac{\sigma'}{\gamma'}]\!]$
$= [\![u[e_1\langle \vec{w_1}\rangle,\ldots,e_m\langle \vec{w_m}\rangle \,|\, c\langle x_1,\ldots,x_n\rangle \overline{[\Gamma]}] \,|\, \frac{\sigma'}{\gamma}]\!]$

Inductive Case: Application
$[\![(s\,t)\{c\langle x_1,\ldots,x_n\rangle\}_e \,|\, \frac{\sigma}{\gamma}]\!] = [\![s\{c\langle x_1,\ldots,x_n\rangle\}_e \,|\, \frac{\sigma}{\gamma}]\!]\,[\![t\{c\langle x_1,\ldots,x_n\rangle\}_e \,|\, \frac{\sigma}{\gamma}]\!]$
$\overset{\text{I.H.}}{=} [\![s \,|\, \frac{\sigma'}{\gamma}]\!]\,[\![t \,|\, \frac{\sigma'}{\gamma}]\!] = [\![s\,t \,|\, \frac{\sigma'}{\gamma}]\!]$

Inductive Case: Abstraction
$[\![(z\langle z\rangle.t)\{c\langle x_1,\ldots,x_n\rangle\}_e \,|\, \frac{\sigma}{\gamma}]\!] = \lambda z.[\![t\{c\langle x_1,\ldots,x_n\rangle\}_e \,|\, \frac{\sigma}{\gamma}]\!]$
$\overset{\text{I.H.}}{=} \lambda z.[\![t \,|\, \frac{\sigma'}{\gamma}]\!] = [\![z\langle z\rangle.t \,|\, \frac{\sigma'}{\gamma}]\!]$

Inductive Case: Phantom-Abstraction
$[\![(d\langle z_1,\ldots,z_m\rangle.t)\{c\langle x_1,\ldots,x_n\rangle\}_e \,|\, \frac{\sigma}{\gamma}]\!] = \lambda d.[\![t\{c\langle x_1,\ldots,x_n\rangle\}_e \,|\, \frac{\sigma''}{\gamma}]\!]$
$\overset{\text{I.H.}}{=} \lambda d.[\![t \,|\, \frac{\sigma'''}{\gamma}]\!] = [\![d\langle z_1,\ldots,z_m\rangle.t \,|\, \frac{\sigma'}{\gamma}]\!]$

Inductive Case: Sharing
$[\![u[z_1,\ldots,z_m \leftarrow t]\{c\langle x_1,\ldots,x_n\rangle\}_e \,|\, \frac{\sigma}{\gamma}]\!]$
$= [\![u\{c\langle x_1,\ldots,x_n\rangle\}_e[z_1,\ldots,z_m \leftarrow t\{c\langle x_1,\ldots,x_n\rangle\}_e] \,|\, \frac{\sigma}{\gamma}]\!]$
$= [\![u\{c\langle x_1,\ldots,x_n\rangle\}_e \,|\, \frac{\sigma''}{\gamma}]\!] \overset{\text{I.H.}}{=} [\![u \,|\, \frac{\sigma'''}{\gamma}]\!] = [\![u[z_1,\ldots,z_m \leftarrow t] \,|\, \frac{\sigma'}{\gamma}]\!]$

Inductive Case: Distributor
$[\![u[e_1\langle \vec{w_1}\rangle,\ldots,e_m\langle \vec{w_m}\rangle \,|\, d\langle d\rangle \overline{[\Gamma]}]\{c\langle x_1,\ldots,x_n\rangle\}_e \,|\, \frac{\sigma}{\gamma}]\!]$
$= [\![u[e_1\langle \vec{w_1}\rangle,\ldots,e_m\langle \vec{w_m}\rangle \,|\, d\langle d\rangle \overline{[\Gamma]}\{c\langle x_1,\ldots,x_n\rangle\}_e] \,|\, \frac{\sigma}{\gamma}]\!]$
$= [\![u\overline{[\Gamma]}\{c\langle x_1,\ldots,x_n\rangle\}_e \,|\, \frac{\sigma}{\gamma'}]\!] \overset{\text{I.H.}}{=} [\![u\overline{[\Gamma]} \,|\, \frac{\sigma'}{\gamma'}]\!]$
$= [\![u[e_1\langle \vec{w_1}\rangle,\ldots,e_m\langle \vec{w_m}\rangle \,|\, d\langle d\rangle \overline{[\Gamma]}] \,|\, \frac{\sigma}{\gamma'}]\!]$ □

Now that we have defined the three operations, we can formalise reduction in Λ_a^S.

3.2 Reduction

Reduction for the spinal atomic λ-calculus performs atomic duplication of terms. The reductions will allow to copy the constructors of a term individually rather than the whole complete term. We divide the reductions into four categories, the beta reduction, deletion reductions, duplication reductions and lifting reductions. Beta reduction will follow the

same intuition as the λ-calculus. Deletion reductions will deal with weakenings i.e. $s[\leftarrow t]$. Duplication will deal with sharings (that are not weakenings) and the distributor, and lifting reductions will deal with the scope of closures. The reductions are categorised in this way for the purposes of Chapter 4 and Chapter 5.

3.2.1 Beta reduction

Perhaps the most well known reduction in the λ-calculus is beta reduction i.e. for some $M, N \in \Lambda$ we have $(\lambda x.M)\, N \rightsquigarrow_\beta M\{N/x\}$. This is because the beta step is viewed as a computational step under Curry-Howard interpretation.

Our calculus will have an equivalent rule. It is important to notice that we only allow beta reduction for abstractions and *not* for phantom-abstractions. Phantom-abstractions will first have to be resolved (by eliminating the distributor that binds them) before we can perform a beta step.

$$(x\langle\, x\,\rangle.t)\, s \rightsquigarrow_\beta t\{s/x\} \qquad (\beta)$$

Figure 3.6 shows the graphical interpretation of this rule. The scope (yellow box) of the abstraction is destroyed after reduction, substituting the term s for the variable x.

3.2.2 Deletion

During reduction, it may become necessary to reduce weakened terms i.e. $u[\leftarrow t]$. These can be seen as special cases of the duplication rules (defined in Section 3.2.3). We define these rules seperately for the benefits of Chapter 4 and Chapter 5. The first deletion rule deals with the case where an application is weakened. Figure 3.7a shows the graphical illustration.

$$u[\leftarrow s\, t] \rightsquigarrow_R u[\leftarrow s][\leftarrow t] \qquad (r_1)$$

The next rule deals when a (phantom-)abstraction is weakened, which introduces a distributor that does not bind any phantom-variables. This rule acts the same for both phantom-abstractions and abstractions i.e. when $\vec{x} = c$ and $\vec{x} \neq c$. Figure 3.7b and Figure 3.7c shows the graphical illustration of the rule in the two different cases.

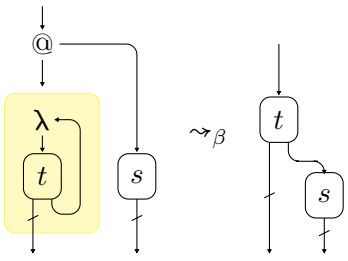


Figure 3.6: Graphical illustration of the beta rule (β)

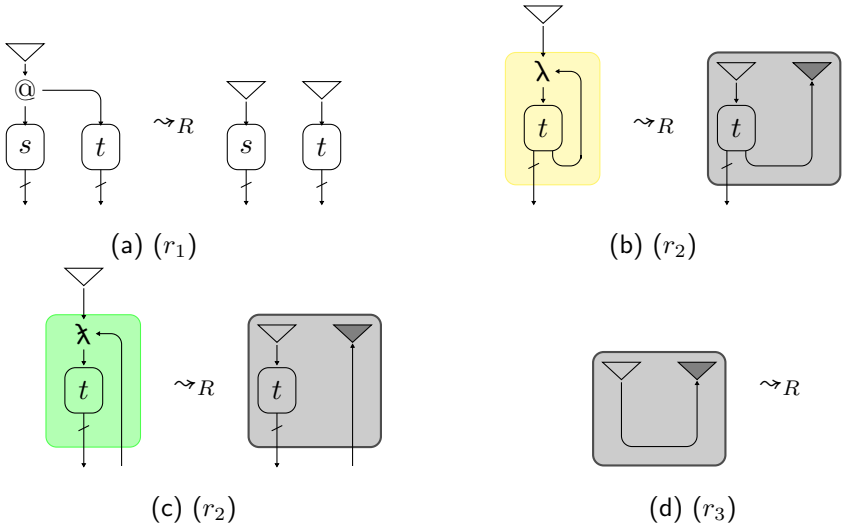


Figure 3.7: Graphical illustrations of deletion rules

$$u[\leftarrow c\langle\,\vec{x}\,\rangle.t] \rightsquigarrow_R u[\,|\,c\langle\,\vec{x}\,\rangle\,[\leftarrow t]] \tag{r_2}$$

Lastly we have the case of eliminating the distributor that does not bind any phantom-variables. This is done by simply removing the distributor. Figure 3.7d shows the graphical illustration for this.

$$u[\,|\,c\langle\,c\,\rangle\,[\leftarrow c]] \rightsquigarrow_R u \tag{r_3}$$

3.2.3 Duplication

The duplication rules are the rules that allow us to duplicate terms atomically, i.e. on individual constructors. When an abstraction or application is shared, we duplicate the constructors independently from all subterms. Each and all introduced variables are fresh.

The first rule is when a sharing meets an application, and works exactly the same as in the original atomic λ-calculus. Figure 3.8a shows the graphical illustration for this rule.

$$u[x_1 \ldots x_n \leftarrow s\,t] \rightsquigarrow_D u\{z_1\,y_1/x_1\}\ldots\{z_n\,y_n/x_n\}[z_1 \ldots z_n \leftarrow s][y_1 \ldots y_n \leftarrow t] \tag{d_1}$$

When an (phantom-)abstraction is duplicated, we introduce the distributor. This distributor captures the phantom-abstractions (the phantom-variables associated to the phantom-abstractions) which are substituted into context, replacing the variables the sharing captured. This rule can be applied to both abstractions and phantom-abstractions, i.e. the variables $\vec{y} = c$ have no effect. Figure 3.8b shows the graphical illustration for when an abstraction is duplicated and Figure 3.8c for a phantom-abstraction.

$$\begin{aligned} &u[x_1,\ldots,x_n \leftarrow c\langle\,\vec{y}\,\rangle.t] \rightsquigarrow_D \\ &\quad u\{e_i\langle\,w_1^i\,\rangle.w_1^i/x_i\}_{1\le i\le n}[e_1\langle\,w_1^1\,\rangle\ldots e_n\langle\,w_1^n\,\rangle\,|\,c\langle\,\vec{y}\,\rangle\,[w_1^1,\ldots,w_1^n \leftarrow t]] \end{aligned} \tag{d_2}$$

Lastly we have the rule that allows us to eliminate a distributor, and finish successfully duplicating an abstraction i.e. converting phantom-abstractions to abstractions. It is important to notice that we can only eliminate distributors that are duplicating abstractions and *not* phantom-abstractions. Intuitively, if a term t has a subterm where a distributor duplicating a phantom-abstraction, one will have to wait until that phantom-abstraction becomes exorcised before completing the duplication and eliminating the distributor.

$$\begin{aligned} u[e_1\langle\,\vec{w}_1\,\rangle \dots e_n\langle\,\vec{w}_n\,\rangle \,|\, c\langle\,c\,\rangle\,[\vec{w}_1, \dots, \vec{w}_n \leftarrow c]] \\ \rightsquigarrow_D u\{e_1\langle\,\vec{w}_1\,\rangle\}_e \dots \{e_n\langle\,\vec{w}_n\,\rangle\}_e \end{aligned} \tag{d_3}$$

Figure 3.8d shows an illustration of eliminating a distributor, but it does not paint the whole picture. It is also possible that one of the phantom-variables bound to the distributor becomes duplicated again, introducing a second distributor. After eliminating the first distributor, the exorcism in this case would implicitly convert the phantom-abstraction into an abstraction, by connecting the wires in the environment to the cosharing node.

3.2.4 Compound

These reduction rules deal with the compound of sharings. The first rule compounds 2 consecutive sharings, and is the same rule as in the original calculus. Figure 3.9a shows the graphical illustration for this rule.

$$\begin{aligned} u[w_1, \dots, w_m \leftarrow y_i][y_1, \dots, y_n \leftarrow t] \rightsquigarrow_C \\ u[y_1, \dots, y_{i-1}, w_1, \dots, w_m, y_{i+1}, \dots, y_n \leftarrow t] \end{aligned} \tag{c_1}$$

We also have the rule where a sharing can be converted into a substitution if it captures exactly one variable. Figure 3.9b shows the graphical illustration for this.

$$u[x \leftarrow t] \rightsquigarrow_C u\{t/x\} \tag{c_2}$$

3.2.5 Lifting

The last set of rules are the lifting rules, which are required to lift closures out of scopes of abstractions and distributors. These rules move closures towards the outside of a term, including lifting closures outside of the scope of a distributor, which may allow us to eliminate said distributor after lifting the closure by applying rule (d_3).

In the original atomic λ-calculus, the lifting rules were equalities in the graphical illustrations. However, in our illustrations we make the scopes of abstractions and distributors explicit, so they will no longer be considered equal in the graphical setting.

For the following rules, we talk about closures i.e. sharings and distributors. We represent the closure as $[\Gamma]$. We write $([\Gamma])_{fv}$ to mean the variables that appear free in the term(s) in the closure. The following 2 rules are also the same as in the original, that involve lifting closures past applications. These rules are considered an equivalence in the view of the graphical illustrations.

$$s[\Gamma]\,t \rightsquigarrow_L (s\,t)[\Gamma] \tag{l_1}$$

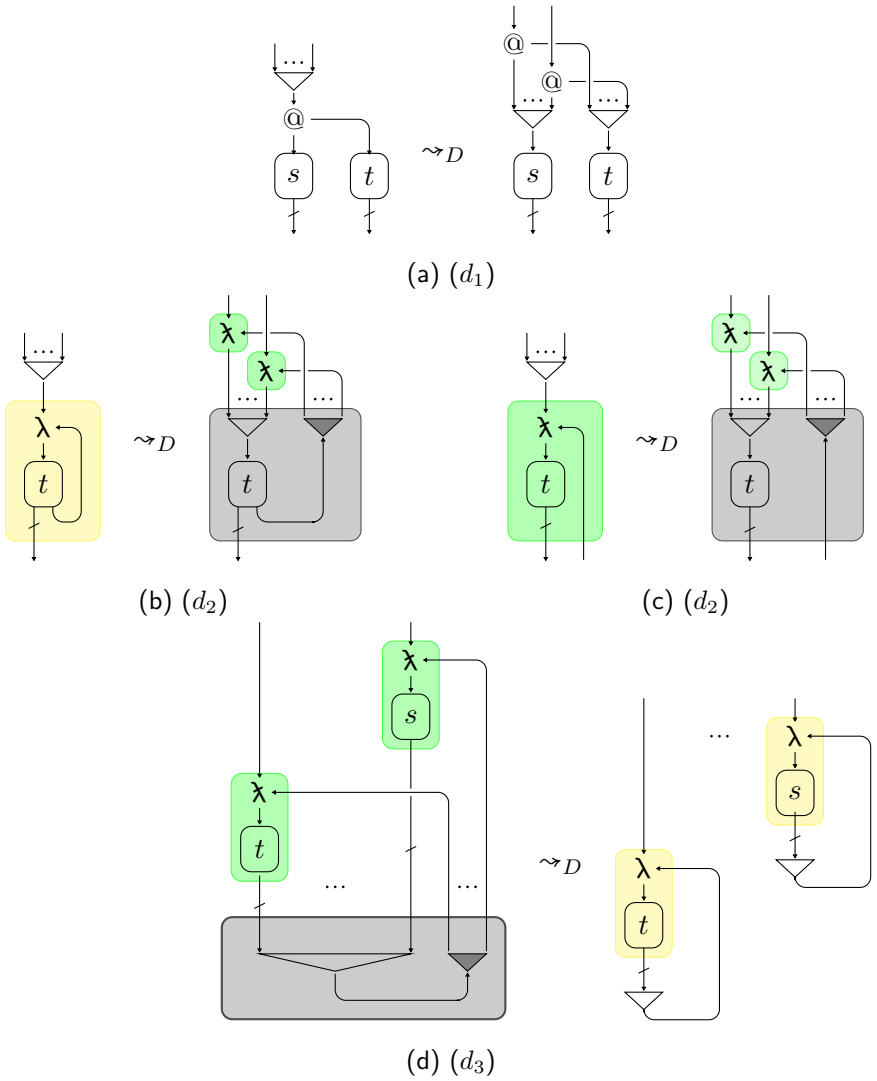
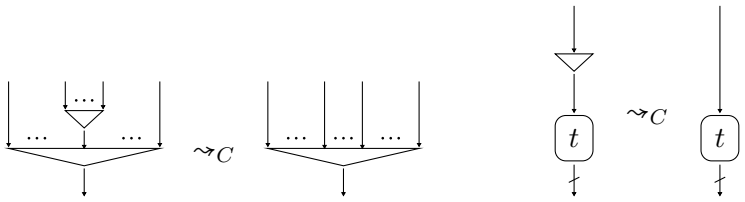


Figure 3.8: Graphical illustrations of duplication rules

$$s\,t[\Gamma] \rightsquigarrow_L (s\,t)[\Gamma] \qquad (l_2)$$

The next rule considers the case of lifting a closure past a (phantom-)abstraction. In the case of lifting past an abstraction, the rule is identical to that of the original. In the case of the phantom-abstraction, we require that each variable in the cover appears free in the body of the term (the same condition as an abstraction). This is to preserve the conditions on terms (Definition 5). Therefore, we also consider that $\vec{x}$ can be equal to d. Figure 3.10a shows the graphical illustration of lifting a sharing past an abstraction and Figure 3.10b shows lifting a distributor past an abstraction.

$$\begin{aligned} &d\langle\,\vec{x}\,\rangle.t[\Gamma] \rightsquigarrow_L (d\langle\,\vec{x}\,\rangle.t)[\Gamma] \\ &\quad \text{iff } x \in \vec{x} \to x \in (t)_{fv} \text{ and } d \notin ([\Gamma])_{fv} \end{aligned} \qquad (l_3)$$



(a) (c_1) (b) (c_2)

Figure 3.9: Graphical illustrations of compound rules

The next rule is again the same as the original equivalent, lifting a closure past a sharing.

$$u[x_1, \ldots, x_n \leftarrow t[\Gamma]] \rightsquigarrow_L u[x_1, \ldots, x_n \leftarrow t][\Gamma] \tag{l_4}$$

The only case left to consider is lifting a closure past a distributor. Just like in the case of abstraction (l_3), the variable occurrence condition must be satisfied i.e. the variables in the cover of the distributor cannot appear in the closure that is being lifted. This rule generates book-keepings that affect the covers captured by the distributor i.e. the variables captured by the closure should not appear in the covers after lifting it out of the distributor. We show the two cases separately to make precise the book-keepings. The fact that the cover of the distributor is an abstraction or a phantom-abstraction does not affect the rule i.e. $\vec{x}$ can be equal to c. Figure 3.10c and Figure 3.10d show the graphical illustrations of these rules where the distributor is duplicating an abstraction.

$$\begin{aligned} &u[e_1\langle\, \vec{w}_1 \,\rangle \ldots e_n\langle\, \vec{w}_n \,\rangle \,|\, c\langle\, \vec{x} \,\rangle\, \overline{[\Gamma]}[\vec{y} \leftarrow t]] \rightsquigarrow_L \\ &u\{(\vec{w}_1/\vec{y})/e_1\}_b \ldots \{(\vec{w}_n/\vec{y})/e_n\}_b[e_1\langle\, \vec{w}_1/\vec{y} \,\rangle \ldots e_n\langle\, \vec{w}_n/\vec{y} \,\rangle \,|\, c\langle\, \vec{x} \,\rangle\, \overline{[\Gamma]}][\vec{y} \leftarrow t] \\ &\qquad\qquad \text{iff } x \in \vec{x} \rightarrow x \notin (t)_{fv} \end{aligned} \tag{l_5}$$

$$\begin{aligned} &u[e_1\langle\, \vec{w}_1 \,\rangle \ldots e_n\langle\, \vec{w}_n \,\rangle \,|\, c\langle\, \vec{x} \,\rangle\, \overline{[\Gamma]}[\overrightarrow{f\langle\, \vec{z} \,\rangle}] \,|\, d\langle\, \vec{a} \,\rangle\, \overline{[\Gamma']}]] \rightsquigarrow_L \\ &u\{(\vec{w}_1/\vec{z})/e_1\}_b \ldots \{(\vec{w}_n/\vec{z})/e_n\}_b[e_1\langle\, \vec{w}_1/\vec{z} \,\rangle \ldots e_n\langle\, \vec{w}_n/\vec{z} \,\rangle \,|\, c\langle\, \vec{x} \,\rangle\, \overline{[\Gamma]}][\overrightarrow{f\langle\, \vec{z} \,\rangle}] \,|\, d\langle\, \vec{a} \,\rangle\, \overline{[\Gamma']}] \\ &\qquad\qquad \text{iff } x \in \vec{x} \rightarrow x \in (u[e_1\langle\, \vec{w}_1 \,\rangle \ldots e_n\langle\, \vec{w}_n \,\rangle \,|\, c\langle\, \vec{x} \,\rangle\, \overline{[\Gamma]}])_{fv} \end{aligned} \tag{l_6}$$

Example 22. *Take the λ-term* $M = (\lambda f.\lambda x.f\,(f\,x))\,\lambda g.\lambda y.g\,(g\,y)$.
Then $(\!|\, M \,|\!) = (f\langle\, f \,\rangle.x\langle\, x \,\rangle.f_1\,(f_2\,x)[f_1, f_2 \leftarrow f])\,(g\langle\, g \,\rangle.y\langle\, y \,\rangle.g_1\,(g_2\,y)[g_1, g_2 \leftarrow g])$.
We then may have the following reduction sequence.

$$\begin{aligned} & (f\langle\, f \,\rangle.x\langle\, x \,\rangle.f_1\,(f_2\,x)[f_1, f_2 \leftarrow f])\,(g\langle\, g \,\rangle.y\langle\, y \,\rangle.g_1\,(g_2\,y)[g_1, g_2 \leftarrow g]) \\ \rightsquigarrow_\beta \quad & x\langle\, x \,\rangle.f_1\,(f_2\,x)[f_1, f_2 \leftarrow g\langle\, g \,\rangle.y\langle\, y \,\rangle.g_1\,(g_2\,y)[g_1, g_2 \leftarrow g]] && (\beta) \\ \rightsquigarrow_D \quad & x\langle\, x \,\rangle.(f_1\langle\, w_1 \,\rangle.w_1\,((f_2\langle\, w_2 \,\rangle.w_2)\,x)) \\ & \quad [f_1\langle\, w_1 \,\rangle, f_2\langle\, w_2 \,\rangle \,|\, g\langle\, g \,\rangle\,[w_1, w_2 \leftarrow y\langle\, y \,\rangle.g_1\,(g_2\,y)[g_1, g_2 \leftarrow g]]] && (d_2) \end{aligned}$$

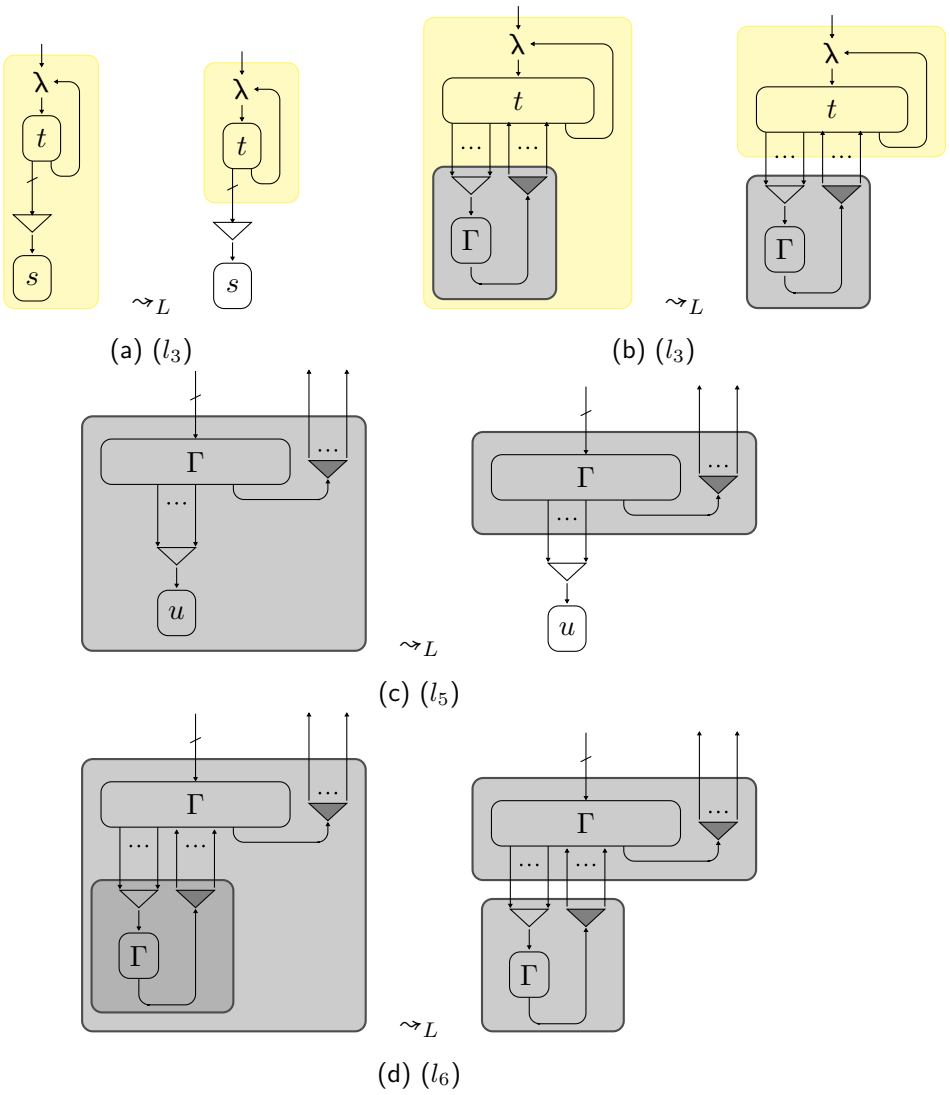


Figure 3.10: Graphical illustrations of lifting rules

$$
\begin{array}{ll}
\rightsquigarrow_D & x\langle x\rangle.(f_1\langle z_1\rangle.y_1\langle z_1\rangle.z_1\,((f_2\langle z_2\rangle.y_2\langle z_2\rangle.z_2)\,x)) \\
& \quad [f_1\langle z_1\rangle, f_2\langle z_2\rangle \mid g\langle g\rangle[y_1\langle z_1\rangle, y_2\langle z_2\rangle \mid y\langle y\rangle \\
& \qquad [z_1, z_2 \leftarrow g_1\,(g_2\,y)[g_1, g_2 \leftarrow g]]]] \qquad (d_2) \\
\rightsquigarrow_L & x\langle x\rangle.(f_1\langle z_1\rangle.y_1\langle z_1\rangle.z_1\,((f_2\langle z_2\rangle.y_2\langle z_2\rangle.z_2)\,x)) \\
& \quad [f_1\langle z_1\rangle, f_2\langle z_2\rangle \mid g\langle g\rangle[y_1\langle z_1\rangle, y_2\langle z_2\rangle \mid y\langle y\rangle \\
& \qquad [z_1, z_2 \leftarrow g_1\,(g_2\,y)][g_1, g_2 \leftarrow g]]] \qquad (l_4) \\
\rightsquigarrow_D & x\langle x\rangle.((f_1\langle a_1, b_1\rangle.y_1\langle a_1, b_1\rangle.a_1\,b_1)\,((f_2\langle a_2, b_2\rangle.y_2\langle a_2, b_2\rangle.a_2\,b_2)\,x)) \\
& \quad [f_1\langle a_1, b_1\rangle, f_2\langle a_2, b_2\rangle \mid g\langle g\rangle[y_1\langle a_1, b_1\rangle, y_2\langle a_2, b_2\rangle \mid y\langle y\rangle \\
& \qquad [a_1, a_2 \leftarrow g_1][b_1, b_2 \leftarrow g_2\,y][g_1, g_2 \leftarrow g]]] \qquad (d_1) \\
\rightsquigarrow_C & x\langle x\rangle.((f_1\langle a_1, b_1\rangle.y_1\langle a_1, b_1\rangle.a_1\,b_1)\,((f_2\langle a_2, b_2\rangle.y_2\langle a_2, b_2\rangle.a_2\,b_2)\,x)) \\
& \quad [f_1\langle a_1, b_1\rangle, f_2\langle a_2, b_2\rangle \mid g\langle g\rangle[y_1\langle a_1, b_1\rangle, y_2\langle a_2, b_2\rangle \mid y\langle y\rangle \\
& \qquad [b_1, b_2 \leftarrow g_2\,y][a_1, a_2, g_2 \leftarrow g]]] \qquad (c_1) \\
\rightsquigarrow_D & x\langle x\rangle.((f_1\langle a_1, b_1, c_1\rangle.y_1\langle a_1, b_1, c_1\rangle.a_1\,(b_1\,c_1)) \\
& \quad ((f_2\langle a_2, b_2, c_2\rangle.y_2\langle a_2, b_2, c_2\rangle.a_2\,(b_2\,c_2))\,x) \\
& \qquad [f_1\langle a_1, b_1, c_1\rangle, f_2\langle a_2, b_2, c_2\rangle \mid g\langle g\rangle[y_1\langle a_1, b_1, c_1\rangle, y_2\langle a_2, b_2, c_2\rangle \mid y\langle y\rangle \\
& \qquad\quad [b_1, b_2 \leftarrow g_2][c_1, c_2 \leftarrow y][a_1, a_2, g_2 \leftarrow g]]] \qquad (d_1) \\
\rightsquigarrow_C & x\langle x\rangle.((f_1\langle a_1, b_1, c_1\rangle.y_1\langle a_1, b_1, c_1\rangle.a_1\,(b_1\,c_1)) \\
& \quad ((f_2\langle a_2, b_2, c_2\rangle.y_2\langle a_2, b_2, c_2\rangle.a_2\,(b_2\,c_2))\,x) \\
& \qquad [f_1\langle a_1, b_1, c_1\rangle, f_2\langle a_2, b_2, c_2\rangle \mid g\langle g\rangle[y_1\langle a_1, b_1, c_1\rangle, y_2\langle a_2, b_2, c_2\rangle \mid y\langle y\rangle \\
& \qquad\quad [c_1, c_2 \leftarrow y][a_1, b_1, a_2, b_2 \leftarrow g]]] \qquad (c_1) \\
\rightsquigarrow_L & x\langle x\rangle.((f_1\langle a_1, b_1, c_1\rangle.y_1\langle c_1\rangle.a_1\,(b_1\,c_1)) \\
& \quad ((f_2\langle a_2, b_2, c_2\rangle.y_2\langle c_2\rangle.a_2\,(b_2\,c_2))\,x) \\
& \qquad [f_1\langle a_1, b_1, c_1\rangle, f_2\langle a_2, b_2, c_2\rangle \mid g\langle g\rangle[y_1\langle c_1\rangle, y_2\langle c_2\rangle \mid y\langle y\rangle \\
& \qquad\quad [c_1, c_2 \leftarrow y]][a_1, b_1, a_2, b_2 \leftarrow g]] \qquad (l_5) \\
\rightsquigarrow_L & x\langle x\rangle.((f_1\langle a_1, b_1\rangle.y_1\langle c_1\rangle.a_1\,(b_1\,c_1))\,((f_2\langle a_2, b_2\rangle.y_2\langle c_2\rangle.a_2\,(b_2\,c_2))\,x)) \\
& \quad [f_1\langle a_1, b_1\rangle, f_2\langle a_2, b_2\rangle \mid g\langle g\rangle\,[a_1, b_1, a_2, b_2 \leftarrow g]] \\
& \qquad [y_1\langle c_1\rangle, y_2\langle c_2\rangle \mid y\langle y\rangle\,[c_1, c_2 \leftarrow y]] \qquad (l_6) \\
\rightsquigarrow_D & x\langle x\rangle.((f_1\langle f_1\rangle.y_1\langle c_1\rangle.a_1\,(b_1\,c_1)[a_1, b_1 \leftarrow f_1]) \\
& \quad ((f_2\langle f_2\rangle.y_2\langle c_2\rangle.a_2\,(b_2\,c_2)[a_2, b_2 \leftarrow f_2])\,x) \\
& \qquad [y_1\langle c_1\rangle, y_2\langle c_2\rangle \mid y\langle y\rangle\,[c_1, c_2 \leftarrow y]] \qquad (d_3) \\
\rightsquigarrow_D & x\langle x\rangle.((f_1\langle f_1\rangle.y_1\langle y_1\rangle.a_1\,(b_1\,y_1)[a_1, b_1 \leftarrow f_1]) \\
& \quad ((f_2\langle f_2\rangle.y_2\langle y_2\rangle.a_2\,(b_2\,y_2)[a_2, b_2 \leftarrow f_2])\,x) \qquad (d_3)
\end{array}
$$

Definition 23. *For a term $t \in \Lambda_a^S$, if there does not exists a term $s \in \Lambda_a^S$ such that $t \rightsquigarrow_{(R,D,L)} s$ then it is said that t is in sharing normal form.*

Lemma 24. *For a $t \in \Lambda_a^S$ in sharing normal form and a $N \in \Lambda$.*

$$[\![\,(\!|\,N\,|\!)\,]\!] = N \qquad (\!|\,[\![\,t\,]\!]\,|\!) = t \qquad \exists_{M \in \Lambda}.t = (\!|\,M\,|\!)$$

Proof. We prove $[\![(\!| N |\!)']\!] = N$ by induction on N

Base Case: Variable
$[\![(\!| x |\!)]\!] = [\![x]\!] = x$

Inductive Case: Application
$[\![(\!| M\, N |\!)]\!] = [\![(\!| M |\!)]\!]\, [\![(\!| N |\!)]\!] = M\, N$

Inductive Case: Abstraction
$[\![(\!| \lambda x.M |\!)]\!]$
Case: $|M|_x = 1$
$= \lambda x.[\![(\!| M |\!)]\!] = \lambda x.M$

Case: $|M|_x = n$
$= \lambda x.[\![(\!| M\frac{n}{x} |\!)'[x_1, \ldots, x_n \leftarrow x]]\!] = \lambda x.[\![(\!| M\frac{n}{x} |\!)' \mid \frac{\sigma}{I}]\!] \overset{\text{prop 18}}{=} \lambda x.[\![(\!| M\frac{n}{x} |\!)']\!]\{x/x_i\}_{1\le i\le n}$
$\overset{\text{I.H.}}{=} \lambda x.M\frac{n}{x}\{x/x_i\}_{1\le i\le n} = \lambda x.M$

We prove $(\!| [\![t]\!] |\!)' = t$ by induction on t

Base Case: Variable
$(\!| [\![x]\!] |\!) = (\!| x |\!) = x$

Inductive Case: Application
$(\!| [\![s\, t]\!] |\!) = (\!| [\![s]\!] |\!)'\, (\!| [\![t]\!] |\!) \overset{\text{I.H.}}{=} s\, t$

Inductive Case: Abstraction
Case: $(\!| [\![x\langle\, x \,\rangle.t]\!] |\!) = x\langle\, x \,\rangle.(\!| [\![t]\!] |\!) \overset{\text{I.H.}}{=} x\langle\, x \,\rangle.t$

Case: $(\!| [\![x\langle\, x \,\rangle.t[x_1, \ldots, x_n \leftarrow x]]\!] |\!) = (\!| \lambda x.[\![t \mid \frac{\sigma}{I}]\!] |\!)$
$\overset{\text{prop 18}}{=} (\!| \lambda x.[\![t]\!]\{x/x_i\}_{1\le i\le n} |\!) = x\langle\, x \,\rangle.(\!| [\![t]\!] |\!)[x_1, \ldots, x_n \leftarrow x]$
$\overset{\text{I.H.}}{=} x\langle\, x \,\rangle.t[x_1, \ldots, x_n \leftarrow x]$ □

Lemma 25 (Sharing reduction preserves denotation)**.** *If* $s \rightsquigarrow_{(R,D,L,C)} t$*, then* $[\![s \mid \frac{\sigma}{\gamma}]\!] = [\![t \mid \frac{\sigma}{\gamma}]\!]$.

Proof. We prove this by induction. First we cover all the base cases.

Case: (r_1)

$$u[\leftarrow s\, t] \rightsquigarrow_R u[\leftarrow s][\leftarrow t]$$

$[\![u[\leftarrow s\, t] \mid \frac{\sigma}{\gamma}]\!] = [\![u \mid \frac{\sigma}{\gamma}]\!] = [\![u[\leftarrow s][\leftarrow t] \mid \frac{\sigma}{\gamma}]\!]$
Case: (r_2)

$$u[\leftarrow c\langle\, \vec{x} \,\rangle.t] \rightsquigarrow_R u[\,|\, c\langle\, \vec{x} \,\rangle\, [\leftarrow t]]$$

$[\![u[\leftarrow c\langle\, \vec{x}\,\rangle.t] \,|\, \frac{\sigma}{\gamma}]\!] = [\![u \,|\, \frac{\sigma}{\gamma}]\!] = [\![u[\,|\, c\langle\, \vec{x}\,\rangle\,[\leftarrow t]] \,|\, \frac{\sigma}{\gamma}]\!]$

Case: (r_3)

$$u[\,|\, c\langle\, c\,\rangle\,[\leftarrow c]] \rightsquigarrow_R u$$

$[\![u[\,|\, c\langle\, c\,\rangle\,[\leftarrow c]] \,|\, \frac{\sigma}{\gamma}]\!] = [\![u \,|\, \frac{\sigma}{\gamma}]\!]$

Case: (c_2)

$$u[x \leftarrow t] \rightsquigarrow_C u\{t/x\}$$

$[\![u[x \leftarrow t] \,|\, \frac{\sigma}{\gamma}]\!] = [\![u \,|\, \frac{\sigma'}{\gamma}]\!] \overset{\text{prop 19}}{=} [\![u\{t/x\} \,|\, \frac{\sigma}{\gamma}]\!]$
where
$\sigma' = \sigma \cup \{x \mapsto [\![t \,|\, \frac{\sigma}{\gamma}]\!]\}$

Case: (d_1)

$$u[x_1 \dots x_n \leftarrow s\,t] \rightsquigarrow_D u\{z_1\,y_1/x_1\} \dots \{z_n\,y_n/x_n\}[z_1 \dots z_n \leftarrow s][y_1 \dots y_n \leftarrow t]$$

$[\![u[x_1 \dots x_n \leftarrow s\,t] \,|\, \frac{\sigma}{\gamma}]\!] = [\![u \,|\, \frac{\sigma'}{\gamma}]\!]$
where
$\sigma' = \sigma \cup \{x_i \mapsto [\![s\,t \,|\, \frac{\sigma}{\gamma}]\!]\}_{1 \le i \le n} = \sigma \cup \{x_i \mapsto [\![s \,|\, \frac{\sigma}{\gamma}]\!]\,[\![t \,|\, \frac{\sigma}{\gamma}]\!]\}_{1 \le i \le n}$

$[\![u\{z_1\,y_1/x_1\} \dots \{z_n\,y_n/x_n\}[z_1 \dots z_n \leftarrow s][y_1 \dots y_n \leftarrow t] \,|\, \frac{\sigma}{\gamma}]\!]$
$= [\![u\{z_1\,y_1/x_1\} \dots \{z_n\,y_n/x_n\} \,|\, \frac{\sigma''}{\gamma}]\!]$
where
$\sigma'' = \sigma \cup \{z_i \mapsto [\![s \,|\, \frac{\sigma}{\gamma}]\!]\}_{1 \le i \le n} \cup \{y_i \mapsto [\![t \,|\, \frac{\sigma}{\gamma}]\!]\}_{1 \le i \le n}$ since $y_i \notin (s)_{fv}$
$\overset{\text{prop 18}}{=} [\![u \,|\, \frac{\sigma'''}{\gamma}]\!]$
where
$\sigma''' = \sigma'' \cup \{x_i \mapsto [\![z_i\,y_i \,|\, \frac{\sigma''}{\gamma}]\!]\}_{1 \le i \le n} = \sigma \cup \{x_i \mapsto [\![s \,|\, \frac{\sigma}{\gamma}]\!]\,[\![t \,|\, \frac{\sigma}{\gamma}]\!]\}_{1 \le i \le n}$
since z_i and $y_i \notin (u)_{fv}$

Case: (d_2)

$$u[x_1, \dots, x_n \leftarrow c\langle\, \vec{y}\,\rangle.t] \rightsquigarrow_D$$
$$u\{e_i\langle\, w_1^i\,\rangle.w_1^i/x_i\}_{1 \le i \le n}[e_1\langle\, w_1^1\,\rangle \dots e_n\langle\, w_1^n\,\rangle \,|\, c\langle\, \vec{y}\,\rangle\,[w_1^1, \dots, w_1^n \leftarrow t]]$$

SubCase: $\vec{y} = c$
$[\![u[x_1, \dots, x_n \leftarrow c\langle\, c\,\rangle.t] \,|\, \frac{\sigma}{\gamma}]\!] = [\![u \,|\, \frac{\sigma'}{\gamma}]\!]$
where $\sigma' = \sigma \cup \{x_i \mapsto [\![c\langle\, c\,\rangle.t \,|\, \frac{\sigma}{\gamma}]\!]\}_{1 \le i \le n} = \sigma \cup \{x_i \mapsto \lambda c.[\![t \,|\, \frac{\sigma}{\gamma}]\!]\}_{1 \le i \le n}$

$[\![u\{e_i\langle\, w_1^i\,\rangle.w_1^i/x_i\}_{1 \le i \le n}[e_1\langle\, w_1^1\,\rangle \dots e_n\langle\, w_1^n\,\rangle \,|\, c\langle\, c\,\rangle\,[w_1^1, \dots, w_1^n \leftarrow t]] \,|\, \frac{\sigma}{\gamma}]\!]$
$[\![u\{e_i\langle\, w_1^i\,\rangle.w_1^i/x_i\}_{1 \le i \le n}[w_1^1, \dots, w_1^n \leftarrow t] \,|\, \frac{\sigma}{\gamma'}]\!]$
$= [\![u\{e_i\langle\, w_1^i\,\rangle.w_1^i/x_i\}_{1 \le i \le n} \,|\, \frac{\sigma'}{\gamma'}]\!]$
where
$\gamma' = \gamma \cup \{e_1 \mapsto c, \dots, e_n \mapsto c\}$

$\sigma' = \sigma \cup \{w_1^i \mapsto [\![\, t \,|\, \frac{\sigma}{\gamma'} \,]\!]\}_{1\le i\le n} = \sigma \cup \{w_1^i \mapsto [\![\, t \,|\, \frac{\sigma}{\gamma} \,]\!]\}_{1\le i\le n}$
$\overset{\text{prop 18}}{=} [\![\, u \,|\, \frac{\sigma''}{\gamma'} \,]\!] = [\![\, u \,|\, \frac{\sigma''}{\gamma} \,]\!]$
where
$\sigma'' = \sigma' \cup \{x_i \mapsto [\![\, e_i\langle\, w_1^i \,\rangle.w_1^i \,|\, \frac{\sigma'}{\gamma'} \,]\!]\}_{1\le i\le n} = \sigma' \cup \{x_i \mapsto \lambda e_i.[\![\, w_1^i \,|\, \frac{\sigma_i'}{\gamma'} \,]\!]\}_{1\le i\le n}$
$= \sigma' \cup \{x_i \mapsto \lambda e_i.[\![\, t \,|\, \frac{\sigma}{\gamma} \,]\!]\{e_i/c\}\}_{1\le i\le n} =_\alpha \sigma' \cup \{x_i \mapsto \lambda c.[\![\, t \,|\, \frac{\sigma}{\gamma} \,]\!]\}_{1\le i\le n}$
$= \sigma \cup \{x_i \mapsto \lambda c.[\![\, t \,|\, \frac{\sigma}{\gamma} \,]\!]\}_{1\le i\le n}$ since $w_1^i \notin (u)_{fv}$

SubCase: $\vec{y} = \{y_1, \ldots, y_m\}$
$[\![\, u[x_1, \ldots, x_n \leftarrow c\langle\, y_1, \ldots, y_m \,\rangle.t] \,|\, \frac{\sigma}{\gamma} \,]\!] = [\![\, u \,|\, \frac{\sigma'}{\gamma} \,]\!]$
where
$\sigma' = \sigma \cup \{x_i \mapsto [\![\, c\langle\, y_1, \ldots, y_m \,\rangle.t \,|\, \frac{\sigma}{\gamma} \,]\!]\}_{1\le i\le n} = \sigma \cup \{x_i \mapsto \lambda c.[\![\, t \,|\, \frac{\sigma''}{\gamma} \,]\!]\}_{1\le i\le n}$
$\sigma'' = \sigma - \{y_1, \ldots, y_m\} \cup \{y_1 \mapsto \sigma(y_1)\{c/\gamma(c)\}, \ldots, y_m \mapsto \sigma(y_m)\{c/\gamma(c)\}\}$

$[\![\, u\{e_i\langle\, w_1^i \,\rangle.w_1^i/x_i\}_{1\le i\le n}[e_1\langle\, w_1^1 \,\rangle \ldots e_n\langle\, w_1^n \,\rangle \,|\, c\langle\, y_1, \ldots, y_m \,\rangle\, [w_1^1, \ldots, w_1^n \leftarrow t]] \,|\, \frac{\sigma}{\gamma} \,]\!]$
$[\![\, u\{e_i\langle\, w_1^i \,\rangle.w_1^i/x_i\}_{1\le i\le n}[w_1^1, \ldots, w_1^n \leftarrow t] \,|\, \frac{\sigma''}{\gamma'} \,]\!]$
where $\gamma' = \gamma \cup \{e_1 \mapsto c, \ldots, e_n \mapsto c\}$
$= [\![\, u\{e_i\langle\, w_1^i \,\rangle.w_1^i/x_i\}_{1\le i\le n} \,|\, \frac{\sigma'''}{\gamma'} \,]\!]$
where $\sigma''' = \sigma'' \cup \{w_1^i \mapsto [\![\, t \,|\, \frac{\sigma''}{\gamma'} \,]\!]\}_{1\le i\le n} = \sigma'' \cup \{w_1^i \mapsto [\![\, t \,|\, \frac{\sigma''}{\gamma} \,]\!]\}_{1\le i\le n}$
$\overset{\text{prop 18}}{=} [\![\, u \,|\, \frac{\sigma''''}{\gamma'} \,]\!] = [\![\, u \,|\, \frac{\sigma''''}{\gamma} \,]\!] = [\![\, u \,|\, \frac{\sigma''}{\gamma} \,]\!]$
where $\sigma'''' = \sigma''' \cup \{x_i \mapsto [\![\, e_i\langle\, w_1^i \,\rangle.w_1^i \,|\, \frac{\sigma'''}{\gamma'} \,]\!]\}_{1\le i\le n} = \sigma''' \cup \{x_i \mapsto \lambda e_i.[\![\, w_1^i \,|\, \frac{\sigma_i'''}{\gamma'} \,]\!]\}_{1\le i\le n}$
$= \sigma''' \cup \{x_i \mapsto \lambda e_i.[\![\, t \,|\, \frac{\sigma''}{\gamma} \,]\!]\{e_i/\gamma'(e_i)\}\}_{1\le i\le n} =_\alpha \sigma''' \cup \{x_i \mapsto \lambda c.[\![\, t \,|\, \frac{\sigma''}{\gamma} \,]\!]\}_{1\le i\le n}$

Case: (d_3)

$$u[e_1\langle\, \vec{w}_1 \,\rangle \ldots e_n\langle\, \vec{w}_n \,\rangle \,|\, c\langle\, c \,\rangle\, [\vec{w}_1, \ldots, \vec{w}_n \leftarrow c]] \rightsquigarrow_D u\{e_1\langle\, \vec{w}_1 \,\rangle\}_e \ldots \{e_n\langle\, \vec{w}_n \,\rangle\}_e$$

$[\![\, u[e_1\langle\, \vec{w}_1 \,\rangle \ldots e_n\langle\, \vec{w}_n \,\rangle \,|\, c\langle\, c \,\rangle\, [\vec{w}_1, \ldots, \vec{w}_n \leftarrow c]] \,|\, \frac{\sigma}{\gamma} \,]\!]$
$= [\![\, u[\vec{w}_1, \ldots, \vec{w}_n \leftarrow c] \,|\, \frac{\sigma}{\gamma'} \,]\!] = [\![\, u \,|\, \frac{\sigma'}{\gamma'} \,]\!]$
$\overset{\text{prop 21}}{=} [\![\, u\{e_1\langle\, \vec{w}_1 \,\rangle\}_e \ldots \{e_n\langle\, \vec{w}_n \,\rangle\}_e \,|\, \frac{\sigma}{\gamma'} \,]\!] = [\![\, u\{e_1\langle\, \vec{w}_1 \,\rangle\}_e \ldots \{e_n\langle\, \vec{w}_n \,\rangle\}_e \,|\, \frac{\sigma}{\gamma} \,]\!]$

Case: (c_1)

$$u[\vec{w} \leftarrow y][\vec{x} \cdot y \leftarrow t] \rightsquigarrow_C u[\vec{x} \cdot \vec{w} \leftarrow t]$$

$[\![\, u[\vec{w} \leftarrow y][\vec{x} \cdot y \leftarrow t] \,|\, \frac{\sigma}{\gamma} \,]\!] = [\![\, u[\vec{w} \leftarrow y] \,|\, \frac{\sigma'}{\gamma} \,]\!] = [\![\, u \,|\, \frac{\sigma''}{\gamma} \,]\!] = [\![\, u[\vec{x} \cdot \vec{w} \leftarrow t] \,|\, \frac{\sigma}{\gamma} \,]\!]$
where
$\sigma' = \sigma \cup \{x \mapsto [\![\, t \,|\, \frac{\sigma}{\gamma} \,]\!]\}_{\forall x \in \vec{x}} \cup \{y \mapsto [\![\, t \,|\, \frac{\sigma}{\gamma} \,]\!]\}$
$\sigma'' = \sigma' \cup \{w \mapsto [\![\, t \,|\, \frac{\sigma}{\gamma} \,]\!]\}_{\forall w \in \vec{w}}$

For the remaining cases, we say $[\![\, t[\Gamma] \,|\, \frac{\sigma}{\gamma} \,]\!]$ produces $[\![\, t \,|\, \frac{\sigma_\Gamma}{\gamma_\Gamma} \,]\!]$ where σ_Γ and γ_Γ are the resulting maps from interpreting the closure $[\Gamma]$
Case: (l_1)

$$s[\Gamma]\, t \rightsquigarrow_L (s\, t)[\Gamma]$$

$[\![s[\Gamma]\, t \,|\, \frac{\sigma}{\gamma}]\!] = [\![s \,|\, \frac{\sigma_\Gamma}{\gamma_\Gamma}]\!] [\![t \,|\, \frac{\sigma}{\gamma}]\!] = [\![s \,|\, \frac{\sigma_\Gamma}{\gamma_\Gamma}]\!] [\![t \,|\, \frac{\sigma_\Gamma}{\gamma_\Gamma}]\!] = [\![(s\, t)[\Gamma] \,|\, \frac{\sigma}{\gamma}]\!]$

Case: (l_2)

$$s[\Gamma]\, t \rightsquigarrow_L (s\, t)[\Gamma]$$

$[\![s\,(t[\Gamma]) \,|\, \frac{\sigma}{\gamma}]\!] = [\![s \,|\, \frac{\sigma}{\gamma}]\!] [\![t \,|\, \frac{\sigma_\Gamma}{\gamma_\Gamma}]\!] = [\![s \,|\, \frac{\sigma_\Gamma}{\gamma_\Gamma}]\!] [\![t \,|\, \frac{\sigma_\Gamma}{\gamma_\Gamma}]\!] = [\![(s\, t)[\Gamma] \,|\, \frac{\sigma}{\gamma}]\!]$

Case: (l_3)

$$d\langle\, \vec{x} \,\rangle.t[\Gamma] \rightsquigarrow_L (d\langle\, \vec{x} \,\rangle.t)[\Gamma]$$

SubCase: $\vec{x} = d$

$[\![d\langle\, d \,\rangle.t[\Gamma] \,|\, \frac{\sigma}{\gamma}]\!] = \lambda d.[\![t[\Gamma] \,|\, \frac{\sigma}{\gamma}]\!] = \lambda d.[\![t \,|\, \frac{\sigma_\Gamma}{\gamma_\Gamma}]\!] = [\![d\langle\, d \,\rangle.t \,|\, \frac{\sigma_\Gamma}{\gamma_\Gamma}]\!] = [\![(d\langle\, d \,\rangle.t)[\Gamma] \,|\, \frac{\sigma}{\gamma}]\!]$

SubCase: $\vec{x} = x_1, \ldots, x_n$

$[\![d\langle\, x_1, \ldots, x_n \,\rangle.t[\Gamma] \,|\, \frac{\sigma}{\gamma}]\!] = \lambda d.[\![t[\Gamma] \,|\, \frac{\sigma'}{\gamma}]\!] = \lambda d.[\![t \,|\, \frac{\sigma'_\Gamma}{\gamma_\Gamma}]\!] = [\![d\langle\, x_1, \ldots, x_n \,\rangle.t \,|\, \frac{\sigma_\Gamma}{\gamma_\Gamma}]\!]$
$= [\![(d\langle\, x_1, \ldots, x_n \,\rangle.t)[\Gamma] \,|\, \frac{\sigma}{\gamma}]\!]$
since we know $x_1, \ldots, x_n \notin ([\Gamma])_{fv}$

Case: (l_4)

$$u[\vec{x} \leftarrow t[\Gamma]] \rightsquigarrow_L u[\vec{x} \leftarrow t][\Gamma]$$

$[\![u[\vec{x} \leftarrow t[\Gamma]] \,|\, \frac{\sigma}{\gamma}]\!] = [\![u \,|\, \frac{\sigma'}{\gamma}]\!] = [\![u \,|\, \frac{\sigma''}{\gamma_\Gamma}]\!] = [\![u[\vec{x} \leftarrow t] \,|\, \frac{\sigma_\Gamma}{\gamma_\Gamma}]\!] = [\![u[\vec{x} \leftarrow t[\Gamma]] \,|\, \frac{\sigma}{\gamma}]\!]$
where
$\sigma' = \sigma \cup \{x \mapsto [\![t[\Gamma] \,|\, \frac{\sigma}{\gamma}]\!]\}_{\forall x \in \vec{x}} = \sigma \cup \{x \mapsto [\![t \,|\, \frac{\sigma_\Gamma}{\gamma_\Gamma}]\!]\}_{\forall x \in \vec{x}}$
$\sigma'' = \sigma_\Gamma \cup \{x \mapsto [\![t \,|\, \frac{\sigma_\Gamma}{\gamma_\Gamma}]\!]\}_{\forall x \in \vec{x}}$

Cases: (l_5) and (l_6)

$$u[e_1\langle\, \vec{w}_1 \,\rangle \ldots e_n\langle\, \vec{w}_n \,\rangle \,|\, c\langle\, \vec{x} \,\rangle\, \overline{[\Gamma]}[\Gamma]] \rightsquigarrow_L$$
$$u\{\vec{z}_1/e_1\}_b \ldots \{\vec{z}_n/e_n\}_b[e_1\langle\, \vec{z}_1 \,\rangle \ldots e_n\langle\, \vec{z}_n \,\rangle \,|\, c\langle\, \vec{x} \,\rangle\, \overline{[\Gamma]}][\Gamma]$$

SubCase: $\vec{x} = c$

$[\![u[e_1\langle\, \vec{w}_1 \,\rangle \ldots e_n\langle\, \vec{w}_n \,\rangle \,|\, c\langle\, c \,\rangle\, \overline{[\Gamma]}[\Gamma]] \,|\, \frac{\sigma}{\gamma}]\!] = [\![u\overline{[\Gamma]}[\Gamma] \,|\, \frac{\sigma}{\gamma'}]\!] = [\![u\overline{[\Gamma]} \,|\, \frac{\sigma_\Gamma}{\gamma'_\Gamma}]\!]$
$\overset{\text{prop 20}}{=} [\![u\overline{[\Gamma]}\{\vec{z}_1/e_1\}_b \ldots \{\vec{z}_n/e_n\}_b \,|\, \frac{\sigma_\Gamma}{\gamma'_\Gamma}]\!] = [\![u\{\vec{z}_1/e_1\}_b \ldots \{\vec{z}_n/e_n\}_b \overline{[\Gamma]} \,|\, \frac{\sigma_\Gamma}{\gamma'_\Gamma}]\!]$
$= [\![u\{\vec{z}_1/e_1\}_b \ldots \{\vec{z}_n/e_n\}_b[e_1\langle\, \vec{z}_1 \,\rangle \ldots e_n\langle\, \vec{z}_n \,\rangle \,|\, c\langle\, c \,\rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma_\Gamma}{\gamma_\Gamma}]\!]$
$= [\![u\{\vec{z}_1/e_1\}_b \ldots \{\vec{z}_n/e_n\}_b[e_1\langle\, \vec{z}_1 \,\rangle \ldots e_n\langle\, \vec{z}_n \,\rangle \,|\, c\langle\, c \,\rangle\, \overline{[\Gamma]}][\Gamma] \,|\, \frac{\sigma}{\gamma}]\!]$

SubCase: $\vec{x} = x_1, \ldots, x_m$

$[\![u[e_1\langle\, \vec{w}_1 \,\rangle \ldots e_n\langle\, \vec{w}_n \,\rangle \,|\, c\langle\, x_1, \ldots, x_m \,\rangle\, \overline{[\Gamma]}[\Gamma]] \,|\, \frac{\sigma}{\gamma}]\!]$
$= [\![u\overline{[\Gamma]}[\Gamma] \,|\, \frac{\sigma'}{\gamma'}]\!] = [\![u\overline{[\Gamma]} \,|\, \frac{\sigma'_\Gamma}{\gamma'_\Gamma}]\!] \overset{\text{prop 20}}{=} [\![u\overline{[\Gamma]}\{\vec{z}_1/e_1\}_b \ldots \{\vec{z}_n/e_n\}_b \,|\, \frac{\sigma_\Gamma}{\gamma'_\Gamma}]\!]$
$= [\![u\{\vec{z}_1/e_1\}_b \ldots \{\vec{z}_n/e_n\}_b \overline{[\Gamma]} \,|\, \frac{\sigma_\Gamma}{\gamma'_\Gamma}]\!]$
$= [\![u\{\vec{z}_1/e_1\}_b \ldots \{\vec{z}_n/e_n\}_b[e_1\langle\, \vec{z}_1 \,\rangle \ldots e_n\langle\, \vec{z}_n \,\rangle \,|\, c\langle\, x_1, \ldots, x_n \,\rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma_\Gamma}{\gamma_\Gamma}]\!]$
$= [\![u\{\vec{z}_1/e_1\}_b \ldots \{\vec{z}_n/e_n\}_b[e_1\langle\, \vec{z}_1 \,\rangle \ldots e_n\langle\, \vec{z}_n \,\rangle \,|\, c\langle\, x_1, \ldots, x_n \,\rangle\, \overline{[\Gamma]}][\Gamma] \,|\, \frac{\sigma}{\gamma}]\!]$

Now we cover the inductive cases.

Inductive Case: Application $s\,t \rightsquigarrow_{(R,D,L,C)} s'\,t$
$[\![\, s\,t \,|\, \frac{\sigma}{\gamma} \,]\!] = [\![\, s \,|\, \frac{\sigma}{\gamma} \,]\!]\,[\![\, t \,|\, \frac{\sigma}{\gamma} \,]\!] \overset{\text{I.H.}}{=} [\![\, s' \,|\, \frac{\sigma}{\gamma} \,]\!]\,[\![\, t \,|\, \frac{\sigma}{\gamma} \,]\!] = [\![\, s'\,t \,|\, \frac{\sigma}{\gamma} \,]\!]$

Inductive Case: Application $s\,t \rightsquigarrow_{(R,D,L,C)} s\,t'$
$[\![\, s\,t \,|\, \frac{\sigma}{\gamma} \,]\!] = [\![\, s \,|\, \frac{\sigma}{\gamma} \,]\!]\,[\![\, t \,|\, \frac{\sigma}{\gamma} \,]\!] \overset{\text{I.H.}}{=} [\![\, s \,|\, \frac{\sigma}{\gamma} \,]\!]\,[\![\, t' \,|\, \frac{\sigma}{\gamma} \,]\!] = [\![\, s\,t' \,|\, \frac{\sigma}{\gamma} \,]\!]$

Inductive Case: Abstraction $x\langle\, x \,\rangle.t \rightsquigarrow_{(R,D,L,C)} x\langle\, x \,\rangle.t'$
$[\![\, x\langle\, x \,\rangle.t \,|\, \frac{\sigma}{\gamma} \,]\!] = \lambda x.[\![\, t \,|\, \frac{\sigma - \{x\}}{\gamma} \,]\!] \overset{\text{I.H.}}{=} \lambda x.[\![\, t' \,|\, \frac{\sigma - \{x\}}{\gamma} \,]\!] = [\![\, x\langle\, x \,\rangle.t \,|\, \frac{\sigma}{\gamma} \,]\!]$

Inductive Case: Phantom-Abstraction $c\langle\, \vec{x} \,\rangle.t \rightsquigarrow_{(R,D,L,C)} c\langle\, \vec{x} \,\rangle.t'$
$[\![\, c\langle\, x_1, \ldots, x_n \,\rangle.t \,|\, \frac{\sigma}{\gamma} \,]\!] = \lambda x.[\![\, t \,|\, \frac{\sigma'}{\gamma} \,]\!] \overset{\text{I.H.}}{=} \lambda x.[\![\, t' \,|\, \frac{\sigma'}{\gamma} \,]\!] = [\![\, c\langle\, x_1, \ldots, x_n \,\rangle.t \,|\, \frac{\sigma}{\gamma} \,]\!]$
where $\sigma' = \sigma - \{x_1, \ldots, x_n\} \cup \{x_i \mapsto \sigma(x_i)\{c/\gamma(c)\}\}_{1 \le i \le n}$

Inductive Case: Sharing $u[\vec{x} \leftarrow t] \rightsquigarrow_{(R,D,L,C)} u'[\vec{x} \leftarrow t]$
$[\![\, u[x_1, \ldots, x_n \leftarrow t] \,|\, \frac{\sigma}{\gamma} \,]\!] = [\![\, u \,|\, \frac{\sigma'}{\gamma} \,]\!] \overset{\text{I.H.}}{=} [\![\, u' \,|\, \frac{\sigma'}{\gamma} \,]\!] = [\![\, u'[x_1, \ldots, x_n \leftarrow t] \,|\, \frac{\sigma}{\gamma} \,]\!]$
where $\sigma' = \sigma - \{x_1, \ldots, x_n\} \cup \{x_i \mapsto [\![\, t \,|\, \frac{\sigma}{\gamma} \,]\!]\}_{1 \le i \le n}$

Inductive Case: Sharing $u[\vec{x} \leftarrow t] \rightsquigarrow_{(R,D,L,C)} u[\vec{x} \leftarrow t']$
$[\![\, u[x_1, \ldots, x_n \leftarrow t] \,|\, \frac{\sigma}{\gamma} \,]\!] = [\![\, u \,|\, \frac{\sigma'}{\gamma} \,]\!] \overset{\text{I.H.}}{=} [\![\, u \,|\, \frac{\sigma''}{\gamma} \,]\!] = [\![\, u[x_1, \ldots, x_n \leftarrow t'] \,|\, \frac{\sigma}{\gamma} \,]\!]$
where $\sigma' = \sigma - \{x_1, \ldots, x_n\} \cup \{x_i \mapsto [\![\, t \,|\, \frac{\sigma}{\gamma} \,]\!]\}_{1 \le i \le n}$
$\sigma'' = \sigma - \{x_1, \ldots, x_n\} \cup \{x_i \mapsto [\![\, t' \,|\, \frac{\sigma}{\gamma} \,]\!]\}_{1 \le i \le n}$

Inductive Case: Distributor $u[\overrightarrow{e\langle\, \vec{w} \,\rangle} \,|\, c\langle\, c \,\rangle\, \overline{[\Gamma]}] \rightsquigarrow_{(R,D,L,C)} u'[\overrightarrow{e\langle\, \vec{w} \,\rangle} \,|\, c\langle\, c \,\rangle\, \overline{[\Gamma]}]$
$[\![\, u[\overrightarrow{e\langle\, \vec{w} \,\rangle} \,|\, c\langle\, c \,\rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma}{\gamma} \,]\!] = [\![\, u\overline{[\Gamma]} \,|\, \frac{\sigma}{\gamma'} \,]\!] = [\![\, u \,|\, \frac{\sigma'}{\gamma''} \,]\!] \overset{\text{I.H.}}{=} [\![\, u' \,|\, \frac{\sigma'}{\gamma''} \,]\!] = [\![\, u'\overline{[\Gamma]} \,|\, \frac{\sigma}{\gamma'} \,]\!]$
$= [\![\, u'[\overrightarrow{e\langle\, \vec{w} \,\rangle} \,|\, c\langle\, c \,\rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma}{\gamma} \,]\!]$
where $\gamma' = \gamma \cup \{e_i \mapsto c\}_{\forall e_i \in \vec{e}}$

Inductive Case: Distributor $u[\overrightarrow{e\langle\, \vec{w} \,\rangle} \,|\, c\langle\, \vec{x} \,\rangle\, \overline{[\Gamma]}] \rightsquigarrow_{(R,D,L,C)} u'[\overrightarrow{e\langle\, \vec{w} \,\rangle} \,|\, c\langle\, \vec{x} \,\rangle\, \overline{[\Gamma]}]$
$[\![\, u[\overrightarrow{e\langle\, \vec{w} \,\rangle} \,|\, c\langle\, \vec{x} \,\rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma}{\gamma} \,]\!] = [\![\, u\overline{[\Gamma]} \,|\, \frac{\sigma'}{\gamma'} \,]\!] = [\![\, u \,|\, \frac{\sigma''}{\gamma''} \,]\!] \overset{\text{I.H.}}{=} [\![\, u' \,|\, \frac{\sigma'}{\gamma''} \,]\!] = [\![\, u'\overline{[\Gamma]} \,|\, \frac{\sigma'}{\gamma'} \,]\!]$
$= [\![\, u'[\overrightarrow{e\langle\, \vec{w} \,\rangle} \,|\, c\langle\, \vec{x} \,\rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma}{\gamma} \,]\!]$

Inductive Case: Distributor $u[\overrightarrow{e\langle\, \vec{x} \,\rangle} \,|\, c\langle\, c \,\rangle\, \overline{[\Gamma]}] \rightsquigarrow_{(R,C,D,L)} u'[\overrightarrow{e\langle\, \vec{x'} \,\rangle} \,|\, c\langle\, c \,\rangle\, \overline{[\Gamma']}]$
$[\![\, u[\overrightarrow{e\langle\, \vec{x} \,\rangle} \,|\, c\langle\, c \,\rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma}{\gamma} \,]\!] = [\![\, u\overline{[\Gamma]} \,|\, \frac{\sigma - \{c\}}{\gamma'} \,]\!] \overset{\text{I.H.}}{=} [\![\, u'\overline{[\Gamma']} \,|\, \frac{\sigma - \{c\}}{\gamma'} \,]\!]$
$= [\![\, u'[\overrightarrow{e\langle\, \vec{x'} \,\rangle} \,|\, c\langle\, c \,\rangle\, \overline{[\Gamma']}] \,|\, \frac{\sigma}{\gamma} \,]\!]$

Inductive Case: Distributor $u[\overrightarrow{e\langle\, \vec{x} \,\rangle} \,|\, c\langle\, \vec{x} \,\rangle\, \overline{[\Gamma]}] \rightsquigarrow_{(R,C,D,L)} u'[\overrightarrow{e\langle\, \vec{x'} \,\rangle} \,|\, c\langle\, \vec{x} \,\rangle\, \overline{[\Gamma']}]$
$[\![\, u[\overrightarrow{e\langle\, \vec{x} \,\rangle} \,|\, c\langle\, \vec{x} \,\rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma}{\gamma} \,]\!] = [\![\, u\overline{[\Gamma]} \,|\, \frac{\sigma'}{\gamma'} \,]\!] \overset{\text{I.H.}}{=} [\![\, u'\overline{[\Gamma']} \,|\, \frac{\sigma'}{\gamma'} \,]\!]$

$= [\![u'[\overrightarrow{e\langle \vec{x'} \rangle} \,|\, c\langle \vec{x} \rangle \overline{[\Gamma']}] \,|\, \frac{\sigma}{\gamma}]\!]$ □

Lemma 26. *Given a term $t \in \Lambda_a^S$, then $(\!|\, [\![t]\!] \,|\!)$ is t in sharing normal form.*

Proof. We can prove this by induction on the longest sharing reduction path from t. Our base case is already covered by Lemma 24. We are then interested in the inductive case, where t is not in sharing normal form. By Lemma 25, $[\![t]\!] = [\![t']\!]$ where $t \rightsquigarrow_{(R,D,L,C)} t'$. By induction hypothesis, $(\!|\, [\![t']\!] \,|\!)$ is in sharing normal form. Hence $(\!|\, [\![t]\!] \,|\!)$ is in sharing normal form. □

3.3 Typing System

We will use here the *open deduction* formalism [GGP10] to type the terms of the spinal atomic λ-calculus. This is the same formalism used to type the atomic λ-calculus of Gundersen, Heijltjes and Parigot [GHP13] and He's atomic $\lambda\mu$-calculus [He18]. The typing system used here is an extension of that of [GHP13], where we restrict the abstraction inference rule to an axiom and add a switch inference rule, which is a common occurrence in systems of deep inference. This small change is exactly what unlocks the ability of spinal duplication; explained in more detail towards the end of this section.

3.3.1 Inference Rules

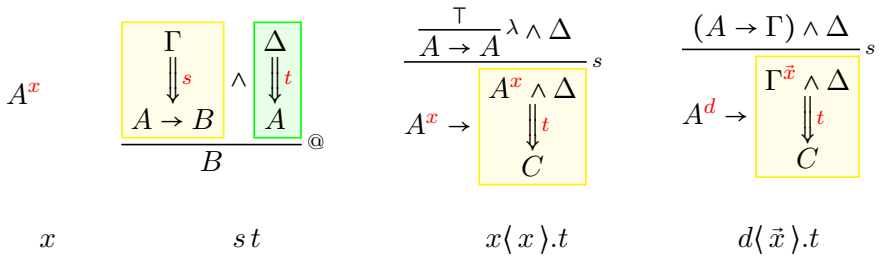
The addition of the switch rule means we can restrict the abstraction rule to an axiom.

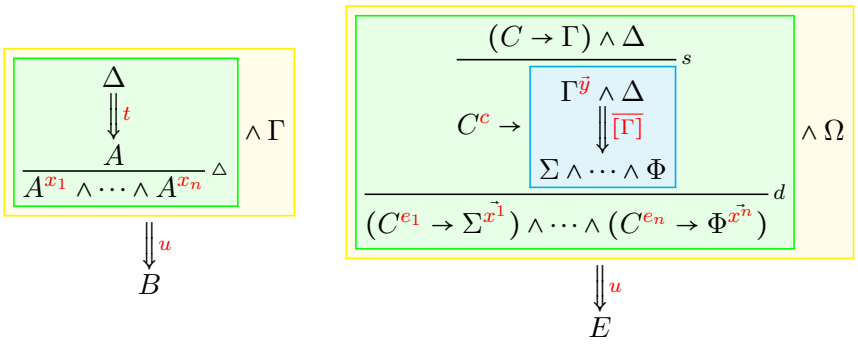
Definition 27. *The typing rules in open deduction for Λ_a^S-terms*

$$\frac{\top}{A \to A}\,\lambda \qquad \frac{(A \to B) \wedge A}{B}\,@ \qquad \frac{A}{A \wedge \cdots \wedge A}\,\triangle$$

$$\frac{(A \to B) \wedge C}{A \to (B \wedge C)}\,s \qquad \frac{A \to (B_1 \wedge \cdots \wedge B_n)}{(A \to B_1) \wedge \cdots \wedge (A \to B_n)}\,d$$

Definition 28. *Typing derivations for Λ_a^S-terms*



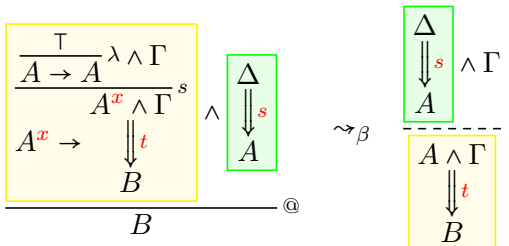


$$u[x_1,\ldots,x_n \leftarrow t] \qquad u[e_1\langle\, x_1^1 \ldots x_{k_1}^1 \,\rangle \ldots e_n\langle\, x_1^n \ldots x_{k_n}^n \,\rangle \,|\, c\langle\, \vec{y} \,\rangle\, \overline{[\Gamma]}]$$

3.3.2 Reduction Rules

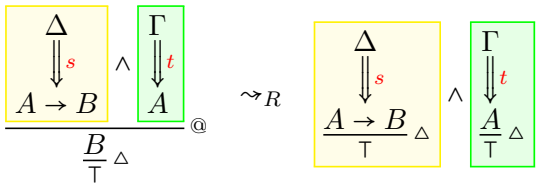
Reduction rules in our calculus correspond to the following transformations in open deduction.

Beta Rule

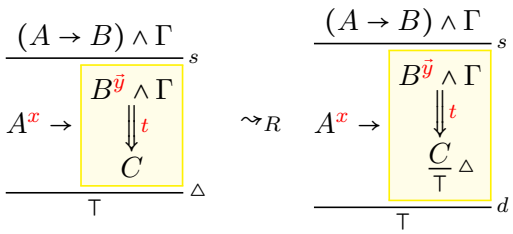


$$(x\langle\, x \,\rangle.t)\, s \qquad \rightsquigarrow_\beta \qquad t\{s/x\} \tag{β}$$

Deletion Rule



$$u[\leftarrow s\, t] \qquad \rightsquigarrow_R \qquad u[\leftarrow s][\leftarrow t] \tag{r_1}$$



$$u[\leftarrow x\langle\, \vec{y} \,\rangle.t] \qquad \rightsquigarrow_R \qquad u[\,|\, x\langle\, \vec{y} \,\rangle\, [\leftarrow t]]$$

(r_2)

$$\dfrac{\dfrac{}{\top}\lambda}{A^x \to \dfrac{A^x}{\top}\triangle}\ d \quad \rightsquigarrow_R \quad \top$$

$$u[\,|\,x\langle\,x\,\rangle\,[\leftarrow x]] \quad \rightsquigarrow_R \quad u$$

(r_3)

Compound Rules

$$\dfrac{\begin{array}{c}\Gamma\\ \Downarrow t\\ A\end{array}}{\vec{A}^{\vec{x}} \wedge \dfrac{A^y}{\vec{A}^{\vec{w}}}\triangle \wedge \vec{A}^{\vec{z}}}\triangle \quad \rightsquigarrow_C \quad \dfrac{\begin{array}{c}\Gamma\\ \Downarrow t\\ A\end{array}}{\vec{A}^{\vec{x}} \wedge \vec{A}^{\vec{w}} \wedge \vec{A}^{\vec{z}}}\triangle$$

$$u[\vec{w} \leftarrow y][\vec{x} \cdot y \cdot \vec{z} \leftarrow t] \quad \rightsquigarrow_C \quad u[\vec{x} \cdot \vec{w} \cdot \vec{z} \leftarrow t]$$

(c_1)

$$\begin{array}{c}\Gamma\\ \Downarrow t\\ \dfrac{A}{A^x}\triangle\\ \Downarrow u\\ C\end{array} \quad \rightsquigarrow_C \quad \begin{array}{c}\Gamma\\ \Downarrow t\\ A\\ \Downarrow u\\ C\end{array}$$

$$u[x \leftarrow t] \quad \rightsquigarrow_C \quad u\{t/y\}$$

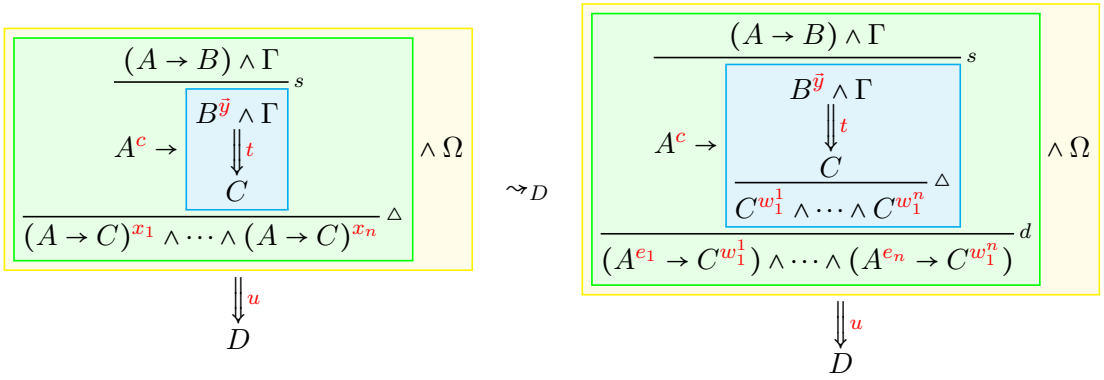
(c_2)

Duplication Rules

$$\begin{array}{c}\dfrac{\dfrac{\begin{array}{c}\Delta\\ \Downarrow s\\ A \to B\end{array} \wedge \begin{array}{c}\Gamma\\ \Downarrow t\\ A\end{array}}{B}@}{B^{x_1} \wedge \cdots \wedge B^{x_n}}\triangle \wedge \Omega\\ \Downarrow u\\ C\end{array} \quad \rightsquigarrow_D \quad \begin{array}{c}\dfrac{\begin{array}{c}\Delta\\ \Downarrow s\\ A \to B\end{array}}{(A \to B)^{z_1} \wedge \cdots \wedge (A \to B)^{z_n}}\triangle \wedge \dfrac{\begin{array}{c}\Gamma\\ \Downarrow t\\ A\end{array}}{A^{y_1} \wedge \cdots \wedge A^{y_n}}\triangle \wedge \Omega\\ \Downarrow u\\ C\end{array}$$

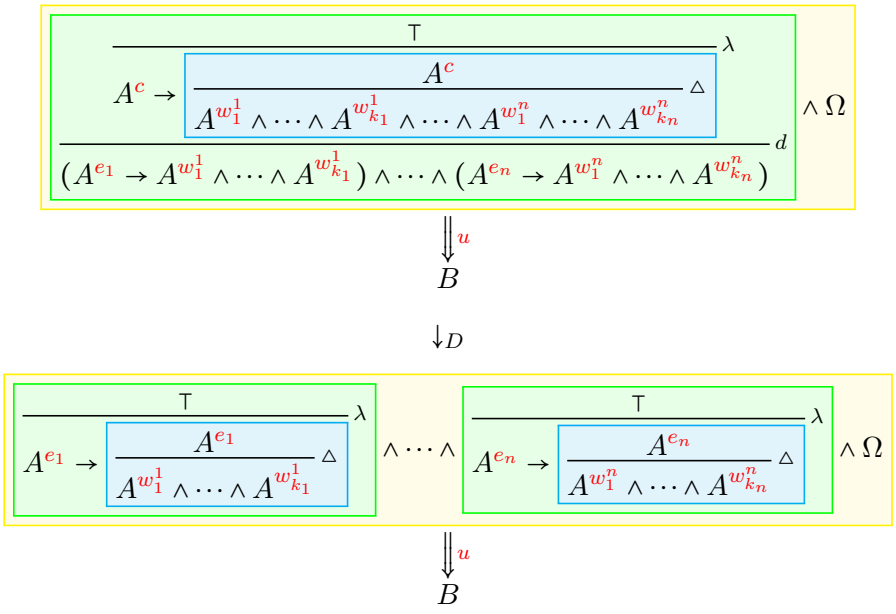
$$u[x_1 \ldots x_n \leftarrow s\,t] \quad \rightsquigarrow_D \quad u\{z_1\,y_1/x_1\} \ldots \{z_n\,y_n/x_n\}[z_1 \ldots z_n \leftarrow s][y_1 \ldots y_n \leftarrow t]$$

(d_1)



$$u[x_1, \ldots, x_n \leftarrow c\langle\, \vec{y} \,\rangle.t] \rightsquigarrow_D$$

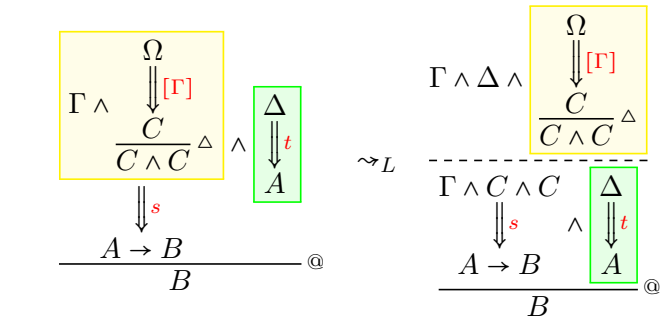
$$u\{e_i\langle\, w_1^i \,\rangle.w_1^i/x_i\}_{1 \le i \le n}[e_1\langle\, w_1^1 \,\rangle \ldots e_n\langle\, w_1^n \,\rangle \,|\, c\langle\, \vec{y} \,\rangle\,[w_1^1, \ldots, w_1^n \leftarrow t]] \tag{d_2}$$



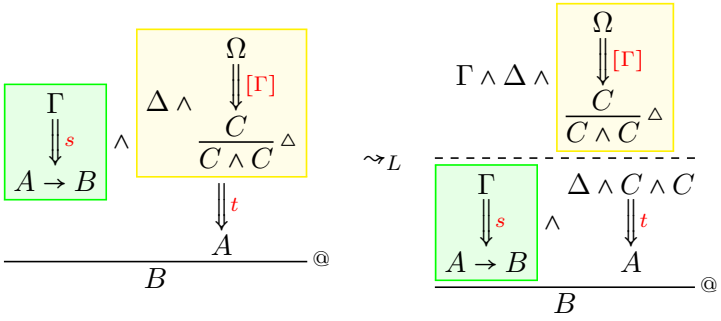
$$u[e_1\langle\, w_1^1, \ldots, w_{k_1}^1 \,\rangle \ldots e_n\langle\, w_1^n, \ldots, w_{k_n}^n \,\rangle \,|\, c\langle\, c \,\rangle\,[w_1^1, \ldots, w_{k_1}^1, \ldots, w_1^n, \ldots, w_{k_n}^n \leftarrow c]] \rightsquigarrow_D$$

$$u\{e_1\langle\, \vec{w}_1 \,\rangle\}_e \ldots \{e_n\langle\, \vec{w}_n \,\rangle\}_e \tag{d_3}$$

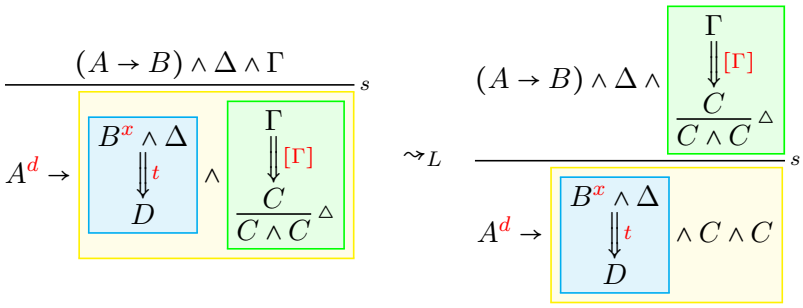
Lifting Rules



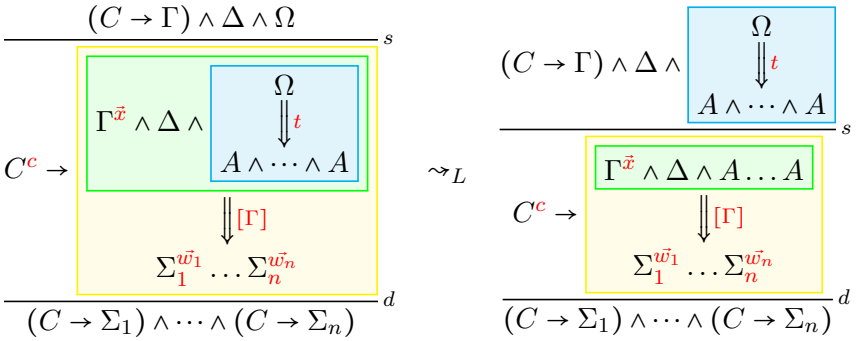
$$s[\Gamma]\,t \quad \rightsquigarrow_L \quad (s\,t)[\Gamma] \qquad (l_1)$$

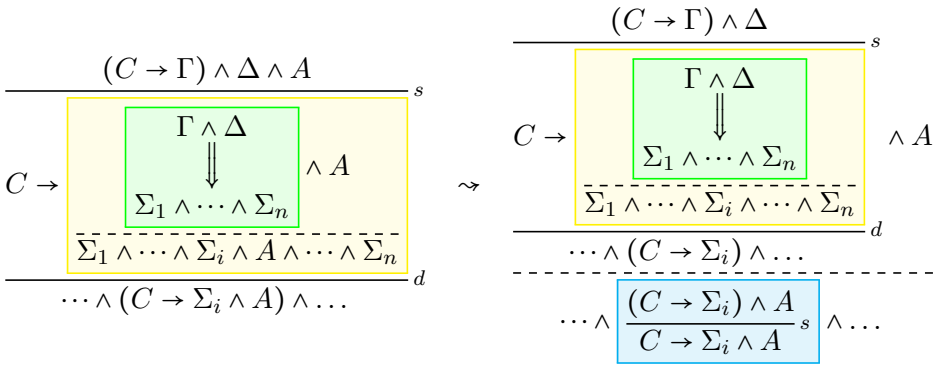


$$s\,t[\Gamma] \quad \rightsquigarrow_L \quad (s\,t)[\Gamma] \qquad (l_2)$$



$$d\langle\, x\,\rangle.t[\Gamma] \quad \rightsquigarrow_L \quad (d\langle\, x\,\rangle.t)[\Gamma] \text{ iff } x \in (t)_{fv} \qquad (l_3)$$





$$u[e_1\langle\,\vec{w}_1\,\rangle \ldots e_n\langle\,\vec{w}_n\,\rangle \,|\, c\langle\,\vec{x}\,\rangle\,[\Gamma][\vec{y} \leftarrow t]] \rightsquigarrow_L$$

$$u\{(\vec{w}_1/\vec{y})/e_1\}_b \ldots \{(\vec{w}_n/\vec{y})/e_n\}_b[e_1\langle\,\vec{w}_1/\vec{y}\,\rangle \ldots e_n\langle\,\vec{w}_n/\vec{y}\,\rangle \,|\, c\langle\,\vec{x}\,\rangle\,[\Gamma]][\vec{y} \leftarrow t]$$

iff all $\vec{x} \notin (t)_{fv}$

(l_5) and (l_6)

The typing derivation for (l_4) remains unchanged, where we only consider the difference of composition similar to (l_1). The last rule shows only the case for (l_5), but covers (l_6) also by replacing the sharing with a distributor. The rule can be seen as two smaller rules combined that represent the rewrite rule, that have a closer relationship with the proof theory rewrite rules. These are

$$u[e_1\langle\,\vec{w}_1\,\rangle \ldots e_n\langle\,\vec{w}_n\,\rangle \,|\, c\langle\,\vec{x}\,\rangle\,[\Gamma][\vec{y} \leftarrow t]] \rightsquigarrow_L$$

$$u[e_1\langle\,\vec{w}_1\,\rangle \ldots e_n\langle\,\vec{w}_n\,\rangle \,|\, c\langle\,\vec{x}\,\rangle\,[\Gamma]][\vec{y} \leftarrow t]$$

iff all $\vec{x} \notin (t)_{fv}$

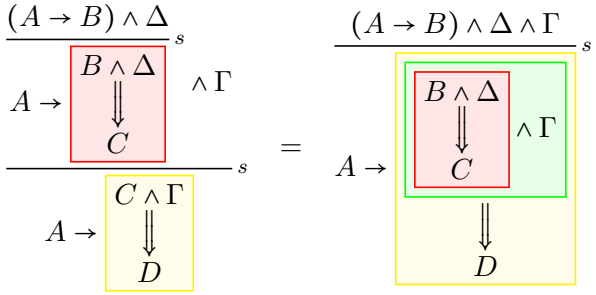
and at most $|\vec{y}|$ applications of the rule

$$u[e\{e_i\langle\,\vec{w} \cdot z\,\rangle\} \,|\, c\langle\,\vec{x}\,\rangle\,[\Gamma][\vec{y} \leftarrow t]] \rightsquigarrow_L$$

$$u\{\vec{w}/e_i\}_b[e\{e_i\langle\,\vec{w}\,\rangle\} \,|\, c\langle\,\vec{x}\,\rangle\,[\Gamma][\vec{y} \leftarrow t]]$$

iff $z \in (u[e\{e_i\langle\,\vec{w} \cdot z\,\rangle\} \,|\, c\langle\,\vec{x}\,\rangle\,[\Gamma][\vec{y} \leftarrow t]])_{fv}$

We can maintain a maximum of one switch rule corresponding with each (phantom-)abstraction with the rewrite rule below. We use this rule to restrict derivations to their canonical form.



Only variables can be brought into scope of a function, not terms. Therefore, unless the derivation is a variable or ends with a sharing inference rule or distribution inference rule, the following relation holds

$$\dfrac{(A \to B) \wedge \Delta \wedge \overset{\Gamma}{\Downarrow} C}{A \to (B \wedge \Delta \wedge C)}\, s \quad = \quad \dfrac{(A \to B) \wedge \Delta \wedge \Gamma}{A \to B \wedge \Delta \wedge \overset{\Gamma}{\Downarrow} C}\, s$$

We discussed how substitutions needed to 'escape' the distributor previously in Section 3.1.2. The following relation shows the logic for this. As the substitution traverses through the distributor, the derivation of the term being substituted is moved from above to below the distribution rule. This was also briefly discussed previously when discussing Figure 3.5.

$$\dfrac{(A \to \Psi) \wedge \Omega}{A^c \to \overset{\Psi^{\vec{y}} \wedge \Omega}{\Downarrow \Gamma} \Gamma_1 \wedge \overset{\Delta^{\vec{z}}}{\Downarrow s} B^x \wedge \Gamma_2}\, s \quad d \qquad = \qquad \dfrac{(A \to \Psi) \wedge \Omega}{A^c \to \overset{\Psi^{\vec{y}} \wedge \Omega}{\Downarrow \Gamma} \Gamma_1 \wedge \Delta^{\vec{z}} \wedge \Gamma_2}\, s \quad d$$

$$\Sigma_1 \wedge (A^{e_i} \to (B^x \wedge \Pi^{\vec{w}})) \wedge \Sigma_2 \qquad\qquad \Sigma_1 \wedge (A^{e_i} \to (\overset{\Delta^{\vec{z}}}{\Downarrow s} B^x \wedge \Pi^{\vec{w}})) \wedge \Sigma_2$$

$$u[e\{e_i\langle\, \vec{w} \cdot x \,\rangle\}\,|\,c\langle\, \vec{y} \,\rangle\,\{s/x\}[\Gamma]] = u\{s/x\}[e\{e_i\langle\, \vec{w} \cdot \vec{z} \,\rangle\}\,|\,c\langle\, \vec{y} \,\rangle\,[\Gamma]]$$

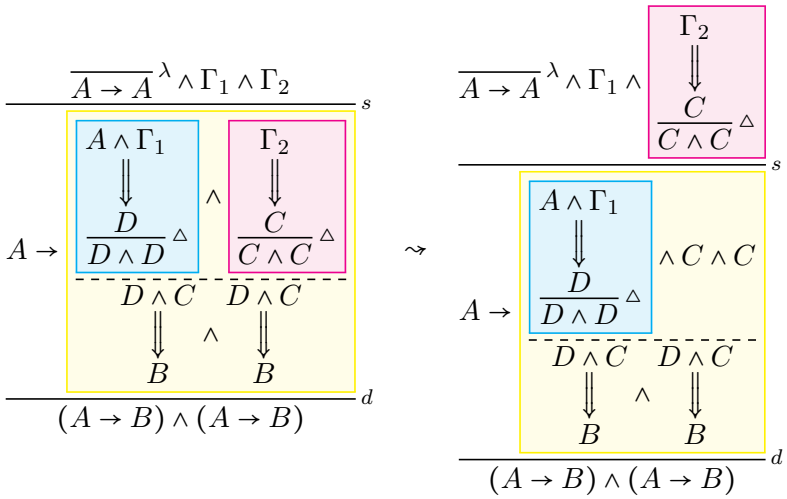
The above demonstrates how reductions are mirrored in the proof system, and thus preserves types. We have the following proposition.

Proposition 29 (Subject Reduction)**.** *If* $s \rightsquigarrow_{(\beta,R,D,L)} t$ *and* $s : \mathcal{T}$*, then* $t : \mathcal{T}$

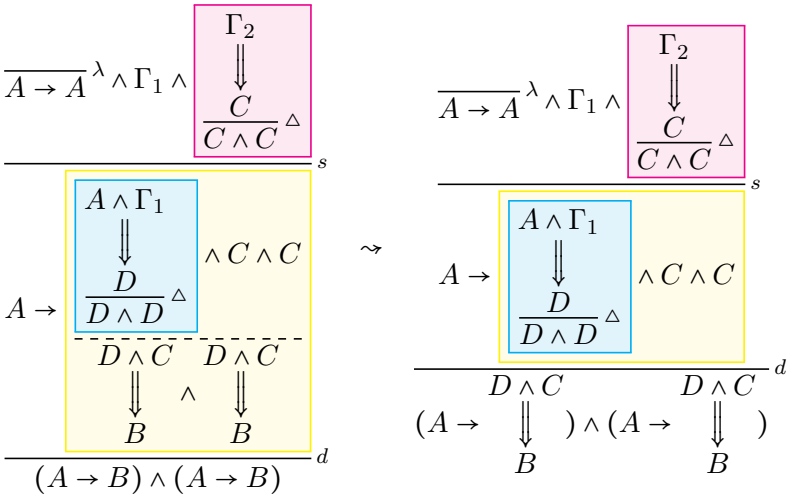
3.3.3 Switch Effects

Chapter 2 discusses the Curry-Howard correspondence between the switch and an end-of-scope operator for the regular λ-calculus. Here we demonstrate how we can take full advantage of the utilities that the switch rule provides, by showing how the use of the switch rule interacts with the distribution rule (medial rule) to allow for more elegant reductions and duplication.

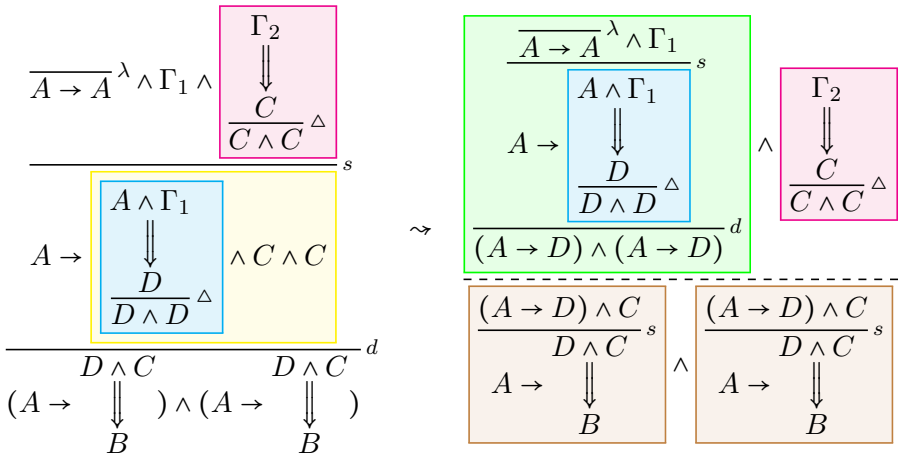
Adding the switch rule to the typing system of the atomic λ-calculus allows for alternative rewrite rules. Because of the switch, we can eliminate distributors independently of each other. As an example, look at the derivation below. We can lift the derivation out of the scope of the abstraction being duplicated. This corresponds to the first proof rewrite rule of (l_5) and (l_6).



Then, we can push the derivations that are a result of duplication underneath the distribution rule. This is what we discussed in Figure 3.5.

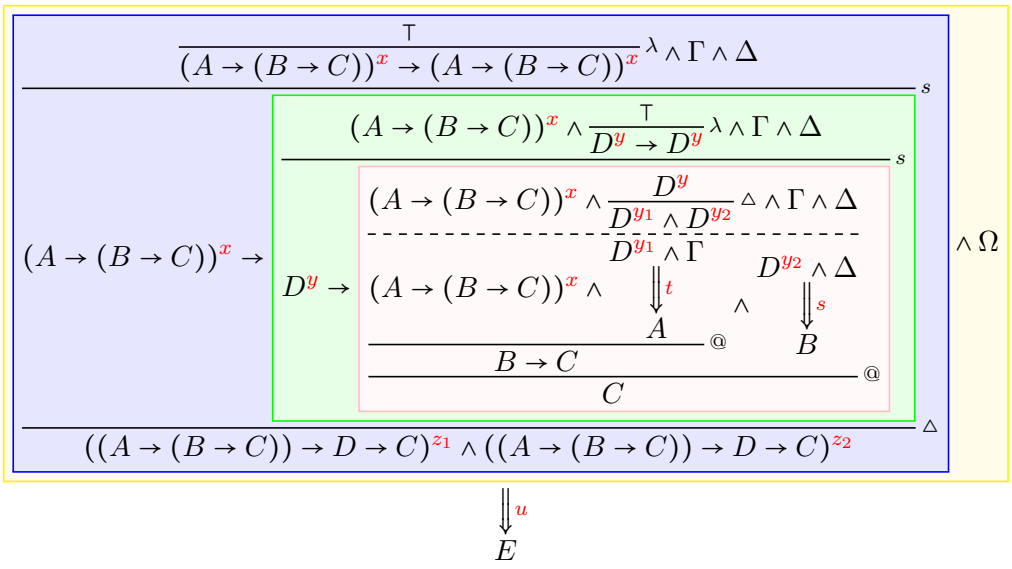


At the moment, these rewrite rules could also have been done without the use of the switch rule. But the addition of the switch rule allows for the following rewrite. Since we are duplicating an abstraction, and we employ the convention that each abstraction has at most one switch rule, it makes sense that at some point we duplicate the switch rules. These correspond to the phantom-abstractions in the calculus.



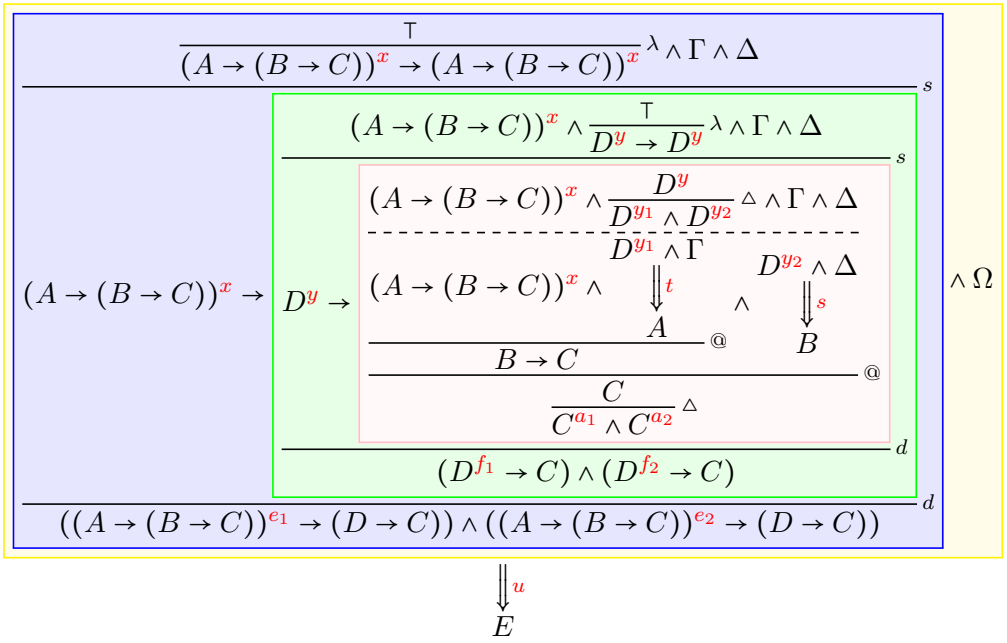
As a larger example, let $r = u[z_1, z_2 \leftarrow \lambda x.\lambda y.(x\,(t\{y_1\}))\,s\{y_2\}[y_1, y_2 \leftarrow y]]$, where t has the variable y_1 occurring in it as a free variable (and the same for s and y_2). In the original atomic λ-calculus reduction would introduce 2 distributors, one for the λx and one for the λy, however, to finish duplicating the λx abstraction i.e. eliminate that distributor, one is forced to first eliminate the distributor associated with λy.

We will discuss an example to demonstrate why our calculus does not have the restriction. We can derive the typing derivation for r as below.



We then perform the reduction (d_1) twice to introduce the distributors in our calculus, resulting in the term

$$u[z_1, z_2 \leftarrow x\langle\,x\,\rangle.y\langle\,y\,\rangle.(x\,(t\{y_1\}))\,s\{y_2\}[y_1, y_2 \leftarrow y]]$$
$$\rightsquigarrow_D^*$$
$$u\{e_1\langle\,a_1\,\rangle.f_1\langle\,a_1\,\rangle.a_1/z_1\}\{e_2\langle\,a_2\,\rangle.f_2\langle\,a_2\,\rangle.a_2/z_2\}$$
$$[e_1\langle\,a_1\,\rangle, e_2\langle\,a_2\,\rangle \,|\, x\langle\,x\,\rangle\,[f_1\langle\,a_1\,\rangle, f_2\langle\,a_2\,\rangle \,|\, y\langle\,y\,\rangle\,[a_1, a_2 \leftarrow (x\,(t\{y_1\}))\,(s\{y_2\})[y_1, y_2 \leftarrow y]]]]$$

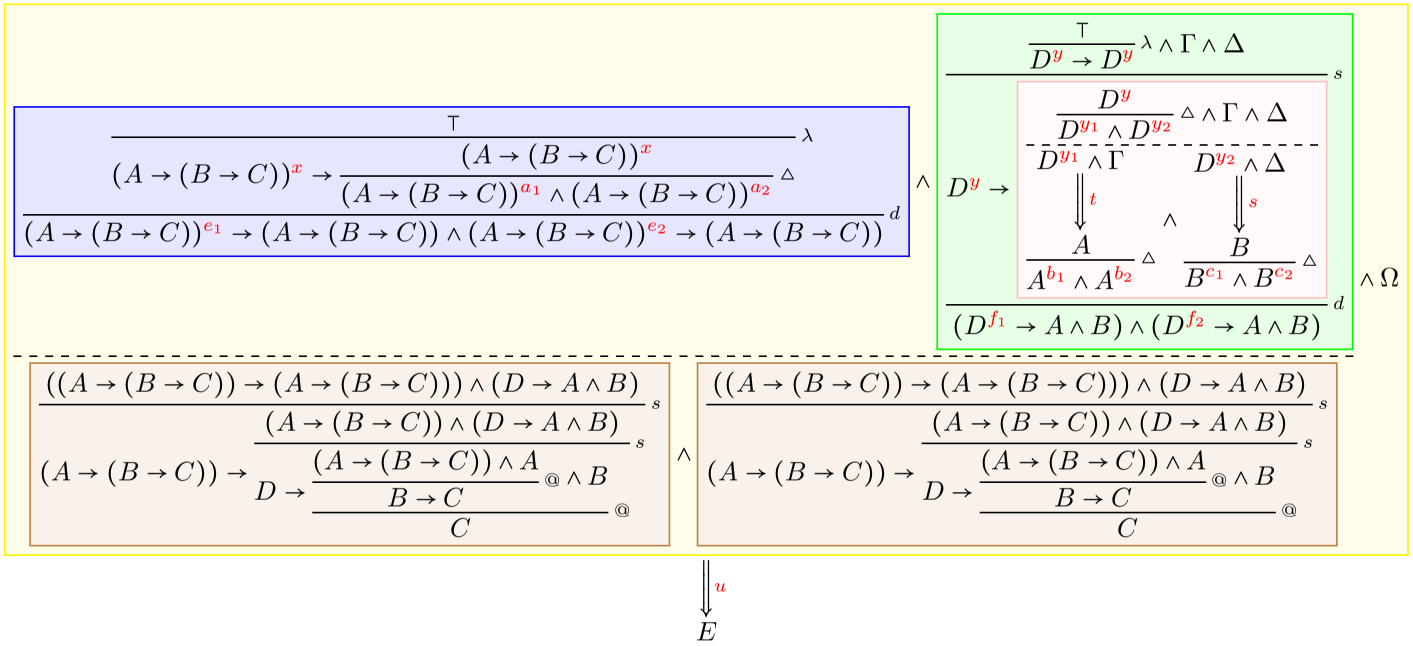


Now we can duplicate the applications using the reduction rule (d_1).

$$
\begin{array}{l}
u\{e_1\langle a_1\rangle.f_1\langle a_1\rangle.a_1/z_1\}\{e_2\langle a_2\rangle.f_2\{a_2\}.a_2/z_2\} \\
\quad [e_1\langle a_1\rangle, e_2\langle a_2\rangle \,|\, x\langle x\rangle\,[f_1\langle a_1\rangle, f_2\langle a_2\rangle \,|\, y\langle y\rangle\,[a_1, a_2 \leftarrow (x\,(t\{y_1\}))\,(s\{y_2\})][y_1, y_2 \leftarrow y]]] \\
\quad \overset{*}{\rightsquigarrow_D} \\
u\{e_1\langle a_1, b_1, c_1\rangle.f_1\langle a_1, b_1, c_1\rangle.(a_1\,b_1)\,c_1/z_1\}\{e_2\langle a_2, b_2, c_2\rangle.f_2\langle a_2, b_2, c_2\rangle.(a_1\,b_1)\,c_2/z_2\} \\
[e_1\langle a_1, b_1, c_1\rangle, e_2\langle a_2, b_2, c_2\rangle \,|\, x\langle x\rangle\,[f_1\langle a_1, b_1, c_1\rangle, f_2\langle a_2, b_2, c_2\rangle \,|\, y\langle y\rangle\,[a_1, a_2 \leftarrow x][b_1, b_2 \leftarrow t\{y_1\}][c_1, c_2 \leftarrow s\{y_2\}][y_1, y_2 \leftarrow y]]]
\end{array}
$$

Then, we can lift our sharing out of the scope of the distributor i.e. the abstraction being duplicated. The reduction rules (l_5) and (l_6) are exactly what allow us to lift the sharings, and thus lift the nested distributor out of the outer distributor, and eliminate the outer distributor before the nested distributor.

$$u\{e_1\langle\, a_1, b_1, c_1\,\rangle.f_1\langle\, a_1, b_1, c_1\,\rangle.(a_1\, b_1)\; c_1/z_1\}\{e_2\langle\, a_2, b_2, c_2\,\rangle.f_2\langle\, a_2, b_2, c_2\,\rangle.(a_1\, b_1)\; c_2/z_2\}$$
$$[e_1\langle\, a_1, b_1, c_1\,\rangle, e_2\langle\, a_2, b_2, c_2\,\rangle \,|\, x\langle\, x\,\rangle\, [f_1\langle\, a_1, b_1, c_1\,\rangle, f_2\langle\, a_2, b_2, c_2\,\rangle \,|\, y\langle\, y\,\rangle\, [a_1, a_2 \leftarrow x][b_1, b_2 \leftarrow t\{y_1\}][c_1, c_2 \leftarrow s\{y_2\}][y_1, y_2 \leftarrow y]]]$$
$$\overset{*}{\rightsquigarrow_L}$$
$$u\{e_1\langle\, a_1\,\rangle.f_1\langle\, b_1, c_1\,\rangle.(a_1\, b_1)\; c_1/z_1\}\{e_2\langle\, a_2\,\rangle.f_2\langle\, b_2, c_2\,\rangle.(a_1\, b_1)\; c_2/z_2\}$$
$$[e_1\langle\, a_1\,\rangle, e_2\langle\, a_2\,\rangle \,|\, x\langle\, x\,\rangle\, [a_1, a_2 \leftarrow x]][f_1\langle\, b_1, c_1\,\rangle, f_2\langle\, b_2, c_2\,\rangle \,|\, y\langle\, y\,\rangle\, [b_1, b_2 \leftarrow t\{y_1\}][c_1, c_2 \leftarrow s\{y_2\}][y_1, y_2 \leftarrow y]]$$



And now we can eliminate the first distributor using (d_3) without continuing to duplicate the terms t and s, thus duplicating less than what the original atomic λ-calculus would duplicate to eliminate this distributor.

Chapter 4

The weakening calculus

删繁就简
shānfán jiùjiǎn
Simplify by cutting out the superfluous

We discuss the weakening calculus, its syntax and reduction rules. Intuitively, the weakening calculus is the original λ-calculus extended with a weakening construct and a *bullet* construct which can be interpretated as a free variable. We later use this calculus to show normalisation of sharing reductions ($\rightsquigarrow_{(R,D,L,C)}$), by formliasing the relationship between the spinal atomic λ-calculus and the weakening calculus.

The weakening calculus is also used to show preservation of strong normalisation with respect to the λ-calculus. A β-step in our calculus may occur within a weakening, and therefore is simulated by zero β-steps in the λ-calculus. Therefore if there is an infinite reduction path located inside a weakening in Λ_a^S, then the reduction path is not preserved in the corresponding λ-term as there are no weakenings. To deal with this, just as done in [AK12a, GHP13, He18], we make use of the weakening calculus. A β-step is non-deleteing precisely because of the weakening construct. If a β-step would be deleting in the λ-calculus, then the weakening calculus would instead keep the deleted term around as 'garbage', which can continue to reduce unless explicitly 'garbage-collected' by extra (non-β) reduction steps. The weakening calculus has already been shown to preserve strong normalisation through the use of a perpetual strategy in [GHP13]. A part of proving PSN is then using the weakening calculus to prove that if $t \in \Lambda_a^S$ has a infinite reduction path, then its translation into the weakening calculus also has an infinite reduction path.

4.1 Syntax and Translations

In this section we introduce the terms of the weakening calculus ($\Lambda_{\mathcal{W}}$), using the same definition as provided in [GHP13].

Definition 30. *The w-terms and the weakening calculus ($\Lambda_{\mathcal{W}}$) are*

$$T, U, V \quad ::= \quad x \quad | \quad \lambda x.T^* \quad | \quad U\,V \quad | \quad T[\leftarrow U] \quad | \quad \bullet$$

(*) where $x \in (T)_{fv}$

The terms provided by Definition 30 are variable, abstraction, application, weakening, and a bullet. In the weakening $T[\leftarrow U]$, the subterm U is *weakened*. The interpretation of atomic terms to weakening terms $[\![- \,|\, \frac{-}{-}]\!]_{\mathcal{W}}$ can be seen as an extension of the translation into the λ-calculus (Definition 13)

Definition 31. *The interpretation $[\![- \,|\, \frac{-}{-}]\!]_{\mathcal{W}} : \Lambda_a^S \times (V \to \Lambda_{\mathcal{W}}) \times (V \to V) \to \Lambda_{\mathcal{W}}$ with maps $\sigma : V \to \Lambda_{\mathcal{W}}$ and $\gamma : V \to V$ is defined as an extension of the translation in (Definition 13) with the following additional special cases.*

$$[\![u[\leftarrow t] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = [\![u \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}[\leftarrow [\![t \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}]$$

$$[\![u[\,|\, c\langle\, c \,\rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = [\![u\overline{[\Gamma]} \,|\, \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}}$$
$$\text{where } \sigma' = \sigma - \{c\} \cup \{c \mapsto \bullet\}$$

$$[\![u[\,|\, c\langle\, x_1, \ldots, x_n \,\rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = [\![u\overline{[\Gamma]} \,|\, \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}}$$
$$\text{where}$$
$$\sigma' = \sigma - \{x_1, \ldots, x_n\} \cup \{x_i \mapsto M_i\{\bullet/\gamma(c)\}\}_{1 \le i \le n}$$

Definition 32. *We say $[\![t]\!]^{\mathcal{W}} = [\![t \,|\, \frac{I}{I}]\!]_{\mathcal{W}}$ where I is the identity function*

Proposition 33. *Given $M \in \Lambda_{\mathcal{W}}$ such that for all $v \in V$, $\gamma(v) \notin (M)_{fv}$ and $\sigma(x) = x$*

$$[\![u \,|\, \frac{\sigma'}{\gamma}]\!] = [\![u \,|\, \frac{\sigma}{\gamma}]\!]\{M/x\}$$

where $\sigma' = (\sigma\{M/x\}) \cup \{x \mapsto M\}$

Proof. We prove this by induction on u. The argument is similar to the proof of Proposition 19. We only discuss here to cases involving the three special cases defined in Definition 31.

Inductive Case: Weakening
$[\![u[\leftarrow t] \,|\, \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}} = [\![u \,|\, \frac{\sigma'}{\gamma}]\!][\leftarrow [\![t \,|\, \frac{\sigma'}{\gamma}]\!]] \overset{\text{I.H.}}{=} [\![u \,|\, \frac{\sigma}{\gamma}]\!]\{M/x\}[\leftarrow [\![t \,|\, \frac{\sigma}{\gamma}]\!]\{M/x\}]$
$= [\![u \,|\, \frac{\sigma}{\gamma}]\!][\leftarrow [\![t \,|\, \frac{\sigma}{\gamma}]\!]]\{M/x\} = [\![u[\leftarrow t] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}\{M/x\}$

Inductive Case: Distributor
$[\![u[\,|\, c\langle\, \vec{x} \,\rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}}$

SubCase: $\vec{x} = c$
$[\![u[\,|\,c\langle\, c\,\rangle\,\overline{[\Gamma]}]\,|\,\frac{\sigma'}{\gamma}\,]\!]_{\mathcal{W}} = [\![u\overline{[\Gamma]}\,|\,\frac{\sigma''}{\gamma}\,]\!]_{\mathcal{W}} \overset{\text{I.H.}}{=} [\![u\overline{[\Gamma]}\,|\,\frac{\sigma'''}{\gamma}\,]\!]_{\mathcal{W}}\{M/x\}$
$= [\![u[\,|\,c\langle\, c\,\rangle\,\overline{[\Gamma]}]\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}}\{M/x\}$
where
$\sigma''' = \sigma \cup \{c \mapsto \bullet\}$
$\sigma'' = \sigma' \cup \{c \mapsto \bullet\}$

SubCase $\vec{x} = x_1, \ldots, x_n$
$[\![u[\,|\,c\langle\, x_1, \ldots, x_n\,\rangle\,\overline{[\Gamma]}]\,|\,\frac{\sigma'}{\gamma}\,]\!]_{\mathcal{W}} = [\![u\overline{[\Gamma]}\,|\,\frac{\sigma''}{\gamma}\,]\!]_{\mathcal{W}} \overset{\text{I.H.}}{=} [\![u\overline{[\Gamma]}\,|\,\frac{\sigma'''}{\gamma}\,]\!]_{\mathcal{W}}\{M/x\}$
$= [\![u[\,|\,c\langle\, c\,\rangle\,\overline{[\Gamma]}]\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}}\{M/x\}$
where
$\sigma' = \sigma_1\{M/x\} \cup \{x_1 \mapsto M_1\{M/x\}, \ldots, x_n \mapsto M_n\{M/x\}\} \cup \{x \mapsto M\}$
$\sigma'' = \sigma_1\{M/x\} \cup \{x_1 \mapsto M_1\{M/x\}\{\bullet/\gamma(c)\}, \ldots, x_n \mapsto M_n\{M/x\}\{\bullet/\gamma(c)\}\} \cup \{x \mapsto M\}$
$\sigma''' = \sigma_1 \cup \{x_1 \mapsto M_1\{\bullet/\gamma(c)\}, \ldots, x_n \mapsto M_n\{\bullet/\gamma(c)\}\}$ □

Proposition 34. *Substitution commutes with the translation in the following way*

$$[\![u\{t/x\}\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}} = [\![u\,|\,\frac{\sigma'}{\gamma}\,]\!]_{\mathcal{W}}$$

where $\sigma' = \sigma \cup \{x \mapsto [\![t\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}}\}$

Proof. We prove this by induction on u. The argument is similar to the proof of Proposition 18. We only discuss here to cases involving the three special cases defined in Definition 31.

Inductive Case: Weakening
$[\![u[\leftarrow s]\{t/x\}\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}} = [\![u\{t/x\}\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}}[\leftarrow [\![s\{t/x\}\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}}]$
$\overset{\text{I.H.}}{=} [\![u\,|\,\frac{\sigma'}{\gamma}\,]\!]_{\mathcal{W}}[\leftarrow [\![s\,|\,\frac{\sigma'}{\gamma}\,]\!]_{\mathcal{W}}] = [\![u[\leftarrow s]\,|\,\frac{\sigma'}{\gamma}\,]\!]_{\mathcal{W}}$

Inductive Case: Distributor
$[\![u[\,|\,c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}]\{t/x\}\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}}$

SubCase: $\vec{x} = c$
$[\![u[\,|\,c\langle\, c\,\rangle\,\overline{[\Gamma]}]\{t/x\}\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}} = [\![u[\,|\,c\langle\, c\,\rangle\,\overline{[\Gamma]}\{t/x\}]\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}}$
$= [\![u\overline{[\Gamma]}\{t/x\}\,|\,\frac{\sigma''}{\gamma'}\,]\!]_{\mathcal{W}} \overset{\text{I.H.}}{=} [\![u\overline{[\Gamma]}\,|\,\frac{\sigma'''}{\gamma'}\,]\!]_{\mathcal{W}} = [\![u[\,|\,c\langle\, c\,\rangle\,\overline{[\Gamma]}]\,|\,\frac{\sigma'}{\gamma}\,]\!]_{\mathcal{W}}$
where
$\sigma'' = \sigma \cup \{c \mapsto \bullet\}$
$\sigma''' = \sigma \cup \{c \mapsto \bullet\} \cup \{x \mapsto [\![t\,|\,\frac{\sigma''}{\gamma'}\,]\!]_{\mathcal{W}}\} = \sigma \cup \{c \mapsto \bullet\} \cup \{x \mapsto [\![t\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}}\}$

SubCase: $\vec{x} = x_1, \ldots, x_n$
$[\![u[\,|\,c\langle\, x_1, \ldots, x_n\,\rangle\,\overline{[\Gamma]}]\{t/x\}\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}}$

SubSubCase: $\vec{x} = x_1, \ldots, x_n, x$
$[\![u[\,|\,c\langle\, x_1, \ldots, x_n, x\,\rangle\,\overline{[\Gamma]}]\{t/x\}\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}}$

$[\![u[\,|\,c\langle\, x_1,\ldots,x_n,y_1,\ldots,y_m \,\rangle\, \overline{[\Gamma]}\{t/x\}]\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}}$
where $\{y_1,\ldots,y_m\} = (t)_{fv}$
$= [\![u\overline{[\Gamma]}\{t/x\}\,|\,\frac{\sigma''}{\gamma}\,]\!]_{\mathcal{W}}$
where
$\sigma = \sigma_1 \cup \{x_1 \mapsto M_1,\ldots,x_n \mapsto M_n, y_1 \mapsto N_1,\ldots,y_m \mapsto N_m\}$
$\sigma'' = \sigma_1 \cup \{x_1 \mapsto M_1\{\bullet/\gamma(c)\},$
$\quad \ldots, x_n \mapsto M_n\{\bullet/\gamma(c)\}, y_1 \mapsto N_1\{\bullet/\gamma(c)\},\ldots,y_m \mapsto N_m\{\bullet/\gamma(c)\}\}$
$\overset{\text{I.H.}}{=} [\![u\overline{[\Gamma]}\,|\,\frac{\sigma'''}{\gamma}\,]\!]_{\mathcal{W}} = [\![u[\,|\,c\langle\, x_1,\ldots,x_n,x \,\rangle\, \overline{[\Gamma]}]\,|\,\frac{\sigma'}{\gamma}\,]\!]_{\mathcal{W}}$
where $\sigma''' = \sigma'' \cup \{x \mapsto [\![t\,|\,\frac{\sigma''}{\gamma}\,]\!]_{\mathcal{W}}\} = \sigma'' \cup \{x \mapsto [\![t\,|\,\frac{\sigma'}{\gamma}\,]\!]_{\mathcal{W}}\{\bullet/\gamma(c)\}\}$
since $\{y_1,\ldots,y_m\} = (t)_{fv}$

SubSubCase: $\vec{x} = x_1,\ldots,x_n$
$[\![u[\,|\,c\langle\, x_1,\ldots,x_n \,\rangle\, \overline{[\Gamma]}]\{t/x\}\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}} = [\![u[\,|\,c\langle\, x_1,\ldots,x_n \,\rangle\, \overline{[\Gamma]}\{t/x\}]\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}}$
$[\![u\overline{[\Gamma]}\{t/x\}\,|\,\frac{\sigma''}{\gamma}\,]\!]_{\mathcal{W}} \overset{\text{I.H.}}{=} [\![u\overline{[\Gamma]}\,|\,\frac{\sigma'''}{\gamma}\,]\!]_{\mathcal{W}} = [\![u[\,|\,c\langle\, x_1,\ldots,x_n \,\rangle\, \overline{[\Gamma]}]\,|\,\frac{\sigma'}{\gamma}\,]\!]_{\mathcal{W}}$
$\sigma = \sigma_1 \cup \{x_1 \mapsto M_1,\ldots,x_n \mapsto M_n\}$
$\sigma'' = \sigma_1 \cup \{x_1 \mapsto M_1\{\bullet/\gamma(c)\},\ldots,x_n \mapsto M_n\{\bullet/\gamma(c)\}\}$
$\sigma''' = \sigma'' \cup \{x \mapsto [\![t\,|\,\frac{\sigma''}{\gamma}\,]\!]_{\mathcal{W}}\} = \sigma'' \cup \{x \mapsto [\![t\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}}\}$
since $\{x_1,\ldots,x_n\} \cap (t)_{fv} = \{\}$ □

Proposition 35. *Book-keeping commutes with the translation in the following way*
if $c\langle\, y_1,\ldots,y_m \,\rangle. \in (u)_{fc}$ *such that* $\{x_1,\ldots,x_n\} \subset \{y_1,\ldots,y_m\}$
and for those $z \in \{y_1,\ldots,y_m\}/\{x_1,\ldots,x_n\}$, $\gamma(c) \notin (\sigma(z))_{fv}$
or if simply $\{x_1,\ldots,x_n\} \cap (u)_{fv} = \{\}$

$$[\![u\{x_1,\ldots,x_n/c\}_b\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}} = [\![u\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}}$$

Proof. We prove this by induction on u. The argument is similar to the proof of Proposition 20. We only discuss here to cases involving the three special cases defined in Definition 31.

Inductive Case: Weakening
$[\![u[\leftarrow t]\{x_1,\ldots,x_n/c\}_b\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}} = [\![u\{x_1,\ldots,x_n/c\}_b\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}}[\leftarrow [\![t\{x_1,\ldots,x_n/c\}_b\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}}]$
$\overset{\text{I.H.}}{=} [\![u\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}}[\leftarrow [\![t\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}}] = [\![u[\leftarrow t]\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}}$

Base Case: Distributor
$[\![u[\,|\,c\langle\, \vec{x} \,\rangle\, \overline{[\Gamma]}]\{x_1,\ldots,x_n/c\}_b\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}} = [\![u[\,|\,c\langle\, x_1,\ldots,x_n \,\rangle\, \overline{[\Gamma]}]\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}}$
$[\![u\overline{[\Gamma]}\,|\,\frac{\sigma'}{\gamma}\,]\!]_{\mathcal{W}} = [\![u[\,|\,c\langle\, \vec{x} \,\rangle\, \overline{[\Gamma]}]\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}}$
where $\sigma = \sigma_1 \cup \{x_1 \mapsto M_1,\ldots,x_n \mapsto M_n\}$
$\sigma' = \sigma_1 \cup \{x_1 \mapsto M_1\{\bullet/\gamma(c)\},\ldots,x_n \mapsto M_n\{\bullet/\gamma(c)\}\}$
and for $x_i \neq y \in \vec{x}$, $\{y \mapsto N\} = \{y \mapsto N\{\bullet/\gamma(c)\}\}$

Inductive Case: Distributor
$[\![u[\,|\,d\langle\, d \,\rangle\, \overline{[\Gamma]}]\{x_1,\ldots,x_n/c\}_b\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}} = [\![u[\,|\,d\langle\, d \,\rangle\, \overline{[\Gamma]}\{x_1,\ldots,x_n/c\}_b]\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}}$
$[\![u\overline{[\Gamma]}\{x_1,\ldots,x_n/c\}_b\,|\,\frac{\sigma'}{\gamma}\,]\!]_{\mathcal{W}} \overset{\text{I.H.}}{=} [\![u\overline{[\Gamma]}\,|\,\frac{\sigma'}{\gamma}\,]\!]_{\mathcal{W}} = [\![u[\,|\,d\langle\, d \,\rangle\, \overline{[\Gamma]}]\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}}$

where $\sigma' = \sigma \cup \{d \mapsto \bullet\}$

$[\![u[\,|\,d\langle\, z_1, \ldots, z_n \,\rangle\, \overline{[\Gamma]}\{x_1, \ldots, x_n/c\}_b \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = [\![u[\,|\,d\langle\, z_1, \ldots, z_n \,\rangle\, \overline{[\Gamma]}\{x_1, \ldots, x_n/c\}_b] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$
$[\![u\overline{[\Gamma]}\{x_1, \ldots, x_n/c\}_b \,|\, \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}} \overset{\text{I.H.}}{=} [\![u\overline{[\Gamma]} \,|\, \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}} = [\![u[\,|\,d\langle\, z_1, \ldots, z_n \,\rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$
where
$\sigma = \sigma_1 \cup \{z_1 \mapsto M_1, \ldots, z_n \mapsto M_n\}$
$\sigma' = \sigma_1 \cup \{z_1 \mapsto M_1\{\bullet/\gamma(d)\}, \ldots, z_n \mapsto M_n\{\bullet/\gamma(d)\}\}$ □

Proposition 36. *Exorcisms commute with the translation in the following way*
if $c\langle\, x_1, \ldots, x_n \,\rangle. \in (u)_{fc}$ *or* $\{x_1, \ldots, x_n\} \cap (u)_{fv} = \{\}$

$$[\![u\{c\langle\, x_1, \ldots, x_n \,\rangle\}_e \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = [\![u \,|\, \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}}$$

where
$\sigma' = \sigma \cup \{x_1 \mapsto c, \ldots, x_n \mapsto c\}$

Proof. We prove this by induction on u. The argument is similar to the proof of Proposition 21. We only discuss here to cases involving the three special cases defined in Definition 31.

Inductive Case: Weakening
$[\![u[\leftarrow t]\{c\langle\, x_1, \ldots, x_n \,\rangle\}_e \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = [\![u\{c\langle\, x_1, \ldots, x_n \,\rangle\}_e \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}[\leftarrow [\![t\{c\langle\, x_1, \ldots, x_n \,\rangle\}_e \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}]$
$\overset{\text{I.H.}}{=} [\![u \,|\, \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}}[\leftarrow [\![t \,|\, \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}}] = [\![u[\leftarrow t] \,|\, \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}}$

Base Case: Distributor
$[\![u[\,|\,c\langle\, x_1, \ldots, x_n \,\rangle\, \overline{[\Gamma]}]\{c\langle\, x_1, \ldots, x_n \,\rangle\}_e \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = [\![u[\,|\,c\langle\, c \,\rangle\, \overline{[\Gamma]}[x_1, \ldots, x_n \leftarrow c]] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$
$= [\![u\overline{[\Gamma]}[x_1, \ldots, x_n \leftarrow c] \,|\, \frac{\sigma''}{\gamma}]\!]_{\mathcal{W}} = [\![u\overline{[\Gamma]} \,|\, \frac{\sigma'''}{\gamma}]\!]_{\mathcal{W}} = [\![u[\,|\,c\langle\, x_1, \ldots, x_n \,\rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}}$
where
$\sigma'' = \sigma \cup \{c \mapsto \bullet\}$
$\sigma''' = \sigma \cup \{x_1 \mapsto \bullet, \ldots, x_n \mapsto \bullet\}$

Inductive Case: Distributor
$[\![u[\,|\,d\langle\, d \,\rangle\, \overline{[\Gamma]}]\{c\langle\, x_1, \ldots, x_n \,\rangle\}_e \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = [\![u[\,|\,d\langle\, d \,\rangle\, \overline{[\Gamma]}\{c\langle\, x_1, \ldots, x_n \,\rangle\}_e] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$
$= [\![u\overline{[\Gamma]}\{c\langle\, x_1, \ldots, x_n \,\rangle\}_e \,|\, \frac{\sigma''}{\gamma}]\!]_{\mathcal{W}} \overset{\text{I.H.}}{=} [\![u\overline{[\Gamma]} \,|\, \frac{\sigma'''}{\gamma}]\!]_{\mathcal{W}} = [\![u[\,|\,d\langle\, d \,\rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}}$
where
$\sigma'' = \sigma \cup \{d \mapsto \bullet\}$
$\sigma''' = \sigma'' \cup \{x_1 \mapsto c, \ldots, x_n \mapsto c\}$

$[\![u[\,|\,d\langle\, z_1, \ldots, z_m \,\rangle\, \overline{[\Gamma]}]\{c\langle\, x_1, \ldots, x_n \,\rangle\}_e \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$
$= [\![u[\,|\,d\langle\, z_1, \ldots, z_m \,\rangle\, \overline{[\Gamma]}\{c\langle\, x_1, \ldots, x_n \,\rangle\}_e] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$
$= [\![u\overline{[\Gamma]}\{c\langle\, x_1, \ldots, x_n \,\rangle\}_e \,|\, \frac{\sigma''}{\gamma}]\!]_{\mathcal{W}} \overset{\text{I.H.}}{=} [\![u\overline{[\Gamma]} \,|\, \frac{\sigma'''}{\gamma}]\!]_{\mathcal{W}} = [\![u[\,|\,d\langle\, d \,\rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}}$
where
$\sigma = \sigma_1 \cup \{z_1 \mapsto N_1, \ldots, z_m \mapsto N_m\}$
$\sigma'' = \sigma_1 \cup \{z_1 \mapsto N_1\{\bullet/\gamma(d)\}, \ldots, z_m \mapsto N_m\{\bullet/\gamma(d)\}\}$
$\sigma''' = \sigma'' \cup \{x_1 \mapsto c, \ldots, x_n \mapsto c\}$ □

We also have translations of the weakening calculus to and from the lambda calculus. Both of these translations have been provided in [GHP13]. We will first discuss the interpretation $(\!|-|\!)^{\mathcal{W}} : \Lambda \to \Lambda_{\mathcal{W}}$.

Definition 37. *The interpretation $M \in \Lambda$, $(\!|-|\!)^{\mathcal{W}} : \Lambda \to \Lambda_{\mathcal{W}}$ is defined by*

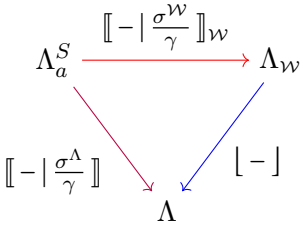
$$
\begin{aligned}
(\!|x|\!)^{\mathcal{W}} &= x \\
(\!|M\,N|\!)^{\mathcal{W}} &= (\!|M|\!)^{\mathcal{W}}\,(\!|N|\!)^{\mathcal{W}} \\
(\!|\lambda x.N|\!)^{\mathcal{W}} &= \begin{cases} \lambda x.(\!|N|\!)^{\mathcal{W}} & \text{if } x \in (N)_{fv} \\ \lambda x.(\!|N|\!)^{\mathcal{W}}[\leftarrow x] & \text{otherwise} \end{cases}
\end{aligned}
$$

The interpretation below is also taken directly from [GHP13]. It should be noted that this is a partial function as the case of interpreting a bullet is undefined. Intuitively this case would be resolved by replacing the bullet with a free fresh variable; resulting in the interpretation to become non-deterministic. We avoid this however by omitting this case since it never arises both in our work nor in the work of [GHP13].

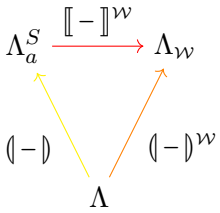
Definition 38. *The partial interpretation $\lfloor - \rfloor : \Lambda_{\mathcal{W}} \to \Lambda$ is defined by*

$$
\begin{aligned}
\lfloor x \rfloor &= x \\
\lfloor U\,V \rfloor &= \lfloor U \rfloor \lfloor V \rfloor \\
\lfloor \lambda x.T \rfloor &= \lambda x.\lfloor T \rfloor \\
\lfloor U[\leftarrow T] \rfloor &= \lfloor U \rfloor
\end{aligned}
$$

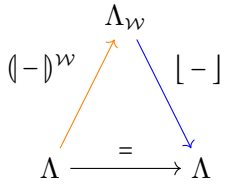
Proposition 39. *For $N \in \Lambda$ and $t \in \Lambda_a^S$ the following properties hold*



$\lfloor [\![t \,|\, \frac{\sigma^{\mathcal{W}}}{\gamma}]\!]_{\mathcal{W}} \rfloor = [\![t \,|\, \frac{\sigma^{\Lambda}}{\gamma}]\!]$



$[\![(\!| N |\!)]\!]^{\mathcal{W}} = (\!| N |\!)^{\mathcal{W}}$



$\lfloor (\!| N |\!)^{\mathcal{W}} \rfloor = N$

where for each $\{x \mapsto M\} \in \sigma^{\mathcal{W}}$ then $\{x \mapsto \lfloor M \rfloor\} \in \sigma^{\Lambda}$

Proof. We prove $\lfloor [\![u \,|\, \frac{\sigma^{\mathcal{W}}}{\gamma}]\!]_{\mathcal{W}} \rfloor = [\![u \,|\, \frac{\sigma^{\Lambda}}{\gamma}]\!]$ by induction on u.

Base Case: Variable
$\lfloor [\![x \,|\, \frac{\sigma^{\mathcal{W}}}{\gamma}]\!]_{\mathcal{W}} \rfloor = \lfloor \sigma^{\mathcal{W}}(x) \rfloor = [\![x \,|\, \frac{\sigma^{\Lambda}}{\gamma}]\!]$

Inductive Case: Application
$\lfloor [\![s\,t \,|\, \frac{\sigma^{\mathcal{W}}}{\gamma}]\!]_{\mathcal{W}} \rfloor = \lfloor [\![s \,|\, \frac{\sigma^{\mathcal{W}}}{\gamma}]\!]_{\mathcal{W}} \rfloor \lfloor [\![t \,|\, \frac{\sigma^{\mathcal{W}}}{\gamma}]\!]_{\mathcal{W}} \rfloor \overset{\text{I.H.}}{=} [\![s \,|\, \frac{\sigma^{\Lambda}}{\gamma}]\!]\, [\![t \,|\, \frac{\sigma^{\Lambda}}{\gamma}]\!] = [\![s\,t \,|\, \frac{\sigma^{\Lambda}}{\gamma}]\!]$

Inductive Case: Abstraction

$\lfloor [\![x\langle x \rangle.t \,|\, \frac{\sigma^{\mathcal{W}}}{\gamma}]\!]_{\mathcal{W}} \rfloor = \lambda x.\lfloor [\![t \,|\, \frac{\sigma^{\mathcal{W}}}{\gamma}]\!]_{\mathcal{W}} \rfloor \overset{\text{I.H.}}{=} \lambda x.[\![t \,|\, \frac{\sigma^{\Lambda}}{\gamma}]\!] = [\![x\langle x \rangle.t \,|\, \frac{\sigma^{\Lambda}}{\gamma}]\!]$

Inductive Case: Phantom-Abstraction
$\lfloor [\![c\langle x_1, \ldots, x_n \rangle.t \,|\, \frac{\sigma^{\mathcal{W}}}{\gamma}]\!]_{\mathcal{W}} \rfloor = \lambda c.\lfloor [\![t \,|\, \frac{\sigma_1^{\mathcal{W}}}{\gamma}]\!]_{\mathcal{W}} \rfloor \overset{\text{I.H.}}{=} \lambda c.[\![x \,|\, \frac{\sigma_1^{\Lambda}}{\gamma}]\!] = [\![c\langle x_1, \ldots, x_n \rangle.t \,|\, \frac{\sigma^{\Lambda}}{\gamma}]\!]$
where
$\sigma^{\mathcal{W}} = \sigma \cup \{x_1 \mapsto M_1, \ldots, x_n \mapsto M_n\}$
$\sigma_1^{\mathcal{W}} = \sigma \cup \{x_1 \mapsto M_1\{c/\gamma(c)\}, \ldots, x_n \mapsto M_n\{c/\gamma(c)\}\}$
$\sigma_1^{\Lambda} = \sigma \cup \{x_1 \mapsto \lfloor M_1 \rfloor\{c/\gamma(c)\}, \ldots, x_n \mapsto \lfloor M_n \rfloor\{c/\gamma(c)\}\}$

Inductive Case: Weakening
$\lfloor [\![u[\leftarrow t] \,|\, \frac{\sigma^{\mathcal{W}}}{\gamma}]\!]_{\mathcal{W}} \rfloor = \lfloor [\![u \,|\, \frac{\sigma^{\mathcal{W}}}{\gamma}]\!]_{\mathcal{W}} \rfloor \overset{\text{I.H.}}{=} [\![u \,|\, \frac{\sigma^{\Lambda}}{\gamma}]\!] = [\![u[\leftarrow t] \,|\, \frac{\sigma^{\Lambda}}{\gamma}]\!]$

Inductive Case: Sharing
$\lfloor [\![u[x_1, \ldots, x_n \leftarrow t] \,|\, \frac{\sigma^{\mathcal{W}}}{\gamma}]\!]_{\mathcal{W}} \rfloor = \lfloor [\![u \,|\, \frac{\sigma_1^{\mathcal{W}}}{\gamma}]\!]_{\mathcal{W}} \rfloor \overset{\text{I.H.}}{=} [\![u \,|\, \frac{\sigma_1^{\Lambda}}{\gamma}]\!] = [\![u[x_1, \ldots, x_n \leftarrow t] \,|\, \frac{\sigma^{\Lambda}}{\gamma}]\!]$
where
$\sigma_1^{\mathcal{W}} = \sigma^{\mathcal{W}} \cup \{x_i \mapsto [\![t \,|\, \frac{\sigma^{\mathcal{W}}}{\gamma}]\!]_{\mathcal{W}}\}_{1 \leq i \leq n}$
$\sigma_1^{\Lambda} = \sigma^{\Lambda} \cup \{x_i \mapsto \lfloor [\![t \,|\, \frac{\sigma^{\mathcal{W}}}{\gamma}]\!]_{\mathcal{W}} \rfloor\}_{1 \leq i \leq n} \overset{\text{I.H.}}{=} \sigma^{\Lambda} \cup \{x_i \mapsto [\![t \,|\, \frac{\sigma^{\Lambda}}{\gamma}]\!]\}_{1 \leq i \leq n}$

Inductive Case: Distributor
$\lfloor [\![u[e_1\langle \vec{w}_1 \rangle, \ldots, e_m\langle \vec{w_m} \rangle \,|\, c\langle \vec{x} \rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma^{\mathcal{W}}}{\gamma}]\!]_{\mathcal{W}} \rfloor$

SubCase: $\vec{x} = c$
$\lfloor [\![u[e_1\langle \vec{w}_1 \rangle, \ldots, e_m\langle \vec{w_m} \rangle \,|\, c\langle c \rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma^{\mathcal{W}}}{\gamma}]\!]_{\mathcal{W}} \rfloor$
$= \lfloor [\![u\overline{[\Gamma]} \,|\, \frac{\sigma}{\gamma'}]\!]_{\mathcal{W}} \rfloor \overset{\text{I.H.}}{=} [\![u\overline{[\Gamma]} \,|\, \frac{\sigma^{\Lambda}}{\gamma'}]\!]$
$= [\![u[e_1\langle \vec{w}_1 \rangle, \ldots, e_m\langle \vec{w_m} \rangle \,|\, c\langle c \rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma^{\Lambda}}{\gamma}]\!]$

SubCase: $\vec{x} = x_1, \ldots, x_n$
$\lfloor [\![u[e_1\langle \vec{w}_1 \rangle, \ldots, e_m\langle \vec{w_m} \rangle \,|\, c\langle x_1, \ldots, x_n \rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma^{\mathcal{W}}}{\gamma}]\!]_{\mathcal{W}} \rfloor$
$\lfloor [\![u\overline{[\Gamma]} \,|\, \frac{\sigma_1^{\mathcal{W}}}{\gamma'}]\!]_{\mathcal{W}} \rfloor \overset{\text{I.H.}}{=} [\![u\overline{[\Gamma]} \,|\, \frac{\sigma_1^{\Lambda}}{\gamma'}]\!]$
$= [\![u[e_1\langle \vec{w}_1 \rangle, \ldots, e_m\langle \vec{w_m} \rangle \,|\, c\langle x_1, \ldots, x_n \rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma^{\Lambda}}{\gamma}]\!]$
where
$\sigma^{\mathcal{W}} = \sigma \cup \{x_1 \mapsto M_1, \ldots, x_n \mapsto M_n\}$
$\sigma_1^{\mathcal{W}} = \sigma \cup \{x_1 \mapsto M_1\{c/\gamma(c)\}, \ldots, x_n \mapsto M_n\{c/\gamma(c)\}\}$
$\sigma_1^{\Lambda} = \sigma \cup \{x_1 \mapsto \lfloor M_1 \rfloor\{c/\gamma(c)\}, \ldots, x_n \mapsto \lfloor M_n \rfloor\{c/\gamma(c)\}\}$

We prove $[\![(\!| N |\!)']\!]^{\mathcal{W}} = (\!| N |\!)^{\mathcal{W}}$ by induction on N. We prove this statement by first proving it for closed terms.

Base Case: Variable
$[\![(\!| x |\!)']\!]^{\mathcal{W}} = [\![x]\!]^{\mathcal{W}} = x = (\!| x |\!)^{\mathcal{W}}$

Inductive Case: Application
$[\![(\!| M\, N |\!)']\!]^{\mathcal{W}} = [\![(\!| M |\!)']\!]^{\mathcal{W}}\, [\![(\!| N |\!)']\!]^{\mathcal{W}} \overset{\text{I.H.}}{=} (\!| M |\!)^{\mathcal{W}}\, (\!| N |\!)^{\mathcal{W}} = (\!| M\, N |\!)^{\mathcal{W}}$

Inductive Case: Abstraction
$[\![(\!| \lambda x.M |\!)']\!]^{\mathcal{W}}$

SubCase: $|M|_x = 0$
$= \lambda x.[\![(\!| M |\!)'[\leftarrow x]]\!]^{\mathcal{W}} = \lambda x.[\![(\!| M |\!)']\!]^{\mathcal{W}}[\leftarrow x] \overset{\text{I.H.}}{=} \lambda x.(\!| M |\!)^{\mathcal{W}}[\leftarrow x] = (\!| \lambda x.M |\!)^{\mathcal{W}}$

SubCase: $|M|_x = 1$
$= \lambda x.[\![(\!| M |\!)']\!]^{\mathcal{W}} \overset{\text{I.H.}}{=} \lambda x.(\!| M |\!)^{\mathcal{W}} = (\!| \lambda x.M |\!)^{\mathcal{W}}$

SubCase: $|M|_x = n > 1$
$= [\![(\!| M\tfrac{n}{x} |\!)'[x^1, \ldots, x^n \leftarrow x]]\!]^{\mathcal{W}} = [\![(\!| M\tfrac{n}{x} |\!)' \mid \tfrac{\sigma}{I}]\!]_{\mathcal{W}} \overset{\text{prop 33}}{=} [\![(\!| M\tfrac{n}{x} |\!)']\!]^{\mathcal{W}}\{x/x_i\}_{1\leq i\leq n}$
$\overset{\text{I.H.}}{=} (\!| M\tfrac{n}{x} |\!)^{\mathcal{W}}\{x/x_i\}_{1\leq i\leq n} = (\!| M |\!)^{\mathcal{W}}$

Now that we have proven is works for closed terms, we can show the statement $[\![(\!| N |\!)]\!]^{\mathcal{W}} = (\!| N |\!)^{\mathcal{W}}$ holds

$[\![(\!| N |\!)]\!]^{\mathcal{W}} = [\![(\!| N\tfrac{n_1}{x_1} \ldots \tfrac{n_k}{x_k} |\!)'[x_1^1, \ldots, x_1^{n_1} \leftarrow x_1] \ldots [x_k^1, \ldots, x_k^{n_k} \leftarrow x_k]]\!]^{\mathcal{W}}$
$\overset{\text{prop 33}}{=} [\![(\!| N\tfrac{n_1}{x_1} \ldots \tfrac{n_k}{x_k} |\!)']\!]^{\mathcal{W}}\{x_i/x_i^j\}_{1\leq i\leq k, 1\leq j\leq n_i} = (\!| N\tfrac{n_1}{x_1} \ldots \tfrac{n_k}{x_k} |\!)^{\mathcal{W}}\{x_i/x_i^j\}_{1\leq i\leq k, 1\leq j\leq n_i} = (\!| N |\!)^{\mathcal{W}}$

□

4.2 Weakening reductions

We wish to use the weakening calculus to show that the spinal atomic λ-calculus satisfies PSN. Observe that β-reduction in the weakening calculus is non-deleting.

Definition 40. *In the weakening calculus, β-reduction is defined as follows, where $\overline{[\Gamma]}$ are weakening constructs.*

$$((\lambda x.T)\overline{[\Gamma]})\,U \to_\beta T\{U/x\}\overline{[\Gamma]} \qquad (w_\beta)$$

Proposition 41. *If $N \in \Lambda$ is strongly normalising, then so is $(\!|\, N \,|\!)^{\mathcal{W}}$*

Proof. A proof for this can be found in [GHP13]. Additionally, similar ideas and results can be found elsewhere, i.e. with memory in [Klo80], the λ-I calculus in [Bar84], the λ-void calculus [AK12a], and the weakening $\lambda\mu$-calculus [He18]. □

When translating from the spinal atomic λ-calculus to the weakening calculus, weakenings are maintained whilst sharings are interpreted through duplication via substitution. Thus the reduction rules in the weakening calculus cover the spinal reductions for nullary distributors and weakenings.

Definition 42. *The weakening reductions ($\to_{\mathcal{W}}$) proceeds as follows.*

$$\lambda x.T[\leftarrow U] \to_{\mathcal{W}} (\lambda x.T)[\leftarrow U] \quad \text{if } x \notin (U)_{fv} \qquad (w_1)$$
$$U[\leftarrow T]\,V \to_{\mathcal{W}} (U\,V)[\leftarrow T] \qquad (w_2)$$
$$U\,V[\leftarrow T] \to_{\mathcal{W}} (U\,V)[\leftarrow T] \qquad (w_3)$$
$$T[\leftarrow U[\leftarrow V]] \to_{\mathcal{W}} T[\leftarrow U][\leftarrow V] \qquad (w_4)$$
$$T[\leftarrow \lambda x.U] \to_{\mathcal{W}} T[\leftarrow U\{\bullet/x\}] \qquad (w_5)$$
$$T[\leftarrow U\,V] \to_{\mathcal{W}} T[\leftarrow U][\leftarrow V] \qquad (w_6)$$
$$T[\leftarrow \bullet] \to_{\mathcal{W}} T \qquad (w_7)$$
$$T[\leftarrow U] \to_{\mathcal{W}} T \quad \text{if } U \text{ is a subterm of } T \qquad (w_8)$$

It is easy to see that these rules correspond to special cases of the sharing reduction rules for Λ_a^S. Lifting a closure relates (w_1) and (l_3), (w_2) and (l_1), (w_3) and (l_2), (w_4) and (l_4), (w_5) and (r_2), and duplicating a term relates (w_6) and (r_1), and (w_7) and (r_3). It is not so obvious to see what the case (w_8) corresponds to. If U is a subterm of T, then in the corresponding Λ_a^S-term this term would be shared and one of the copies would be in a weakening. Thus this reduction relates to the case (c_1), where we remove the weakening. We demonstrate this with the following example.

$$t[\leftarrow y][\vec{x} \cdot y \cdot \vec{z} \leftarrow u] \rightsquigarrow_C t[\vec{x} \cdot \vec{z} \leftarrow u]$$

On the left hand side, the corresponding weakening-term (obtained by $(\!|-|\!)^{\mathcal{W}}$) would have the weakening $[\leftarrow U]$ where $U = (\!|\, u \,|\!)^{\mathcal{W}}$. This is because U is substituted into $[\leftarrow y]$, but on the right hand side this would be gone. This situation can only occur if there are other copies of U substituted into the term. This corresponds to if only the corresponding (c_1) reduction rule can occur.

Lemma 43. *If* $t \rightsquigarrow_{(\beta)} u$ *then* $[\![t]\!]^{\mathcal{W}} \to_{\beta}^{+} [\![u]\!]^{\mathcal{W}}$

Proof. We prove this by induction. Our base case:
$[\![(x\langle x \rangle.t)\, s]\!]^{\mathcal{W}} = (\lambda x.T)\, S = T\{S/x\} = [\![t\{s/x\}]\!]^{\mathcal{W}}$
where $T = [\![t]\!]^{\mathcal{W}}$ and $S = [\![s]\!]^{\mathcal{W}}$

Now we cover the inductive cases.

Inductive Case: Application $s\,t \rightsquigarrow_{(\beta)} s'\,t$
$[\![s\,t \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = [\![s \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}\, [\![t \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} \overset{\text{I.H.}}{\to}_{\mathcal{W}}^{*} [\![s' \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}\, [\![t \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = [\![s'\,t \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$

Inductive Case: Application $s\,t \rightsquigarrow_{(\beta)} s\,t'$
$[\![s\,t \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = [\![s \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}\, [\![t \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} \overset{\text{I.H.}}{\to}_{\mathcal{W}}^{*} [\![s \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}\, [\![t' \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = [\![s\,t' \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$

Inductive Case: Abstraction $x\langle x \rangle.t \rightsquigarrow_{(\beta)} x\langle x \rangle.t'$
$[\![x\langle x \rangle.t \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = \lambda x.[\![t \mid \frac{\sigma - \{x\}}{\gamma}]\!]_{\mathcal{W}} \overset{\text{I.H.}}{\to}_{\mathcal{W}}^{*} \lambda x.[\![t' \mid \frac{\sigma - \{x\}}{\gamma}]\!]_{\mathcal{W}} = [\![x\langle x \rangle.t \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$

Inductive Case: Phantom-Abstraction $c\langle \vec{x} \rangle.t \rightsquigarrow_{(\beta)} c\langle \vec{x} \rangle.t'$
$[\![c\langle x_1, \ldots, x_n \rangle.t \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = \lambda x.[\![t \mid \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}} \overset{\text{I.H.}}{\to}_{\mathcal{W}}^{*} \lambda x.[\![t' \mid \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}} = [\![c\langle x_1, \ldots, x_n \rangle.t \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$
where $\sigma' = \sigma - \{x_1, \ldots, x_n\} \cup \{x_i \mapsto \sigma(x_i)\{c/\gamma(c)\}\}_{1 \le i \le n}$

Inductive Case: Sharing $u[\vec{x} \leftarrow t] \rightsquigarrow_{(\beta)} u'[\vec{x} \leftarrow t]$
$[\![u[x_1, \ldots, x_n \leftarrow t] \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = [\![u \mid \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}} \overset{\text{I.H.}}{\to}_{\mathcal{W}}^{*} [\![u' \mid \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}} = [\![u'[x_1, \ldots, x_n \leftarrow t] \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$
where $\sigma' = \sigma - \{x_1, \ldots, x_n\} \cup \{x_i \mapsto [\![t \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}\}_{1 \le i \le n}$

Inductive Case: Sharing $u[\vec{x} \leftarrow t] \rightsquigarrow_{(\beta)} u[\vec{x} \leftarrow t']$
$[\![u[x_1, \ldots, x_n \leftarrow t] \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = [\![u \mid \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}} \overset{\text{I.H.}}{\to}_{\mathcal{W}}^{*} [\![u \mid \frac{\sigma''}{\gamma}]\!]_{\mathcal{W}} = [\![u[x_1, \ldots, x_n \leftarrow t'] \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$
where $\sigma' = \sigma - \{x_1, \ldots, x_n\} \cup \{x_i \mapsto [\![t \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}\}_{1 \le i \le n}$
$\sigma'' = \sigma - \{x_1, \ldots, x_n\} \cup \{x_i \mapsto [\![t' \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}\}_{1 \le i \le n}$

Inductive Case: Weakening: $u[\leftarrow t] \rightsquigarrow_{(\beta)} u[\leftarrow t']$
$[\![u[\leftarrow t] \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = [\![u \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}[\leftarrow [\![t \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}] \overset{\text{I.H.}}{\to}_{\mathcal{W}}^{*} [\![u \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}[\leftarrow [\![t' \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}] = [\![u[\leftarrow t'] \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$

Inductive Case: Weakening: $u[\leftarrow t] \rightsquigarrow_{(\beta)} u'[\leftarrow t]$
$[\![u[\leftarrow t] \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = [\![u \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}[\leftarrow [\![t \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}] \overset{\text{I.H.}}{\to}_{\mathcal{W}}^{*} [\![u' \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}[\leftarrow [\![t \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}] = [\![u'[\leftarrow t] \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$

Inductive Case: Distributor $u[\overrightarrow{e\langle \vec{w} \rangle} \mid c\langle c \rangle\, \overline{[\Gamma]}] \rightsquigarrow_{(\beta)} u'[\overrightarrow{e\langle \vec{w} \rangle} \mid c\langle c \rangle\, \overline{[\Gamma]}]$
$[\![u[\overrightarrow{e\langle \vec{w} \rangle} \mid c\langle c \rangle\, \overline{[\Gamma]}] \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = [\![u\overline{[\Gamma]} \mid \frac{\sigma}{\gamma'}]\!]_{\mathcal{W}} = [\![u \mid \frac{\sigma'}{\gamma''}]\!]_{\mathcal{W}} \overset{\text{I.H.}}{\to}_{\mathcal{W}}^{*} [\![u' \mid \frac{\sigma'}{\gamma''}]\!]_{\mathcal{W}} = [\![u'\overline{[\Gamma]} \mid \frac{\sigma}{\gamma'}]\!]_{\mathcal{W}}$
$= [\![u'[\overrightarrow{e\langle \vec{w} \rangle} \mid c\langle c \rangle\, \overline{[\Gamma]}] \mid \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$
where $\gamma' = \gamma \cup \{e_i \mapsto c\}_{\forall e_i \in \vec{e}}$

Inductive Case: Distributor $u[\overrightarrow{e\langle\,\vec{w}\,\rangle}\,|\,c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}] \rightsquigarrow_{(\beta)} u'[\overrightarrow{e\langle\,\vec{w}\,\rangle}\,|\,c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}]$
$[\![\, u[\overrightarrow{e\langle\,\vec{w}\,\rangle}\,|\,c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}]\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}} = [\![\, u\overline{[\Gamma]}\,|\,\frac{\sigma'}{\gamma'}\,]\!]_{\mathcal{W}} = [\![\, u\,|\,\frac{\sigma''}{\gamma''}\,]\!]_{\mathcal{W}} \overset{\text{I.H.}}{\to}^*_{\mathcal{W}} [\![\, u'\,|\,\frac{\sigma'}{\gamma''}\,]\!]_{\mathcal{W}} = [\![\, u'\overline{[\Gamma]}\,|\,\frac{\sigma'}{\gamma'}\,]\!]_{\mathcal{W}}$
$= [\![\, u'[\overrightarrow{e\langle\,\vec{w}\,\rangle}\,|\,c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}]\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}}$

Inductive Case: Distributor $u[\overrightarrow{e\langle\,\vec{x}\,\rangle}\,|\,c\langle\,c\,\rangle\,\overline{[\Gamma]}] \rightsquigarrow_{(\beta)} u'[\overrightarrow{e\langle\,\vec{x'}\,\rangle}\,|\,c\langle\,c\,\rangle\,\overline{[\Gamma']}]$
$[\![\, u[\overrightarrow{e\langle\,\vec{x}\,\rangle}\,|\,c\langle\,c\,\rangle\,\overline{[\Gamma]}]\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}} = [\![\, u\overline{[\Gamma]}\,|\,\frac{\sigma-\{c\}}{\gamma'}\,]\!]_{\mathcal{W}} \overset{\text{I.H.}}{\to}^*_{\mathcal{W}} [\![\, u'\overline{[\Gamma']}\,|\,\frac{\sigma-\{c\}}{\gamma'}\,]\!]_{\mathcal{W}}$
$= [\![\, u'[\overrightarrow{e\langle\,\vec{x'}\,\rangle}\,|\,c\langle\,c\,\rangle\,\overline{[\Gamma']}]\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}}$

Inductive Case: Distributor $u[\overrightarrow{e\langle\,\vec{x}\,\rangle}\,|\,c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}] \rightsquigarrow_{(\beta)} u'[\overrightarrow{e\langle\,\vec{x'}\,\rangle}\,|\,c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma']}]$
$[\![\, u[\overrightarrow{e\langle\,\vec{x}\,\rangle}\,|\,c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}]\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}} = [\![\, u\overline{[\Gamma]}\,|\,\frac{\sigma'}{\gamma'}\,]\!]_{\mathcal{W}} \overset{\text{I.H.}}{\to}^*_{\mathcal{W}} [\![\, u'\overline{[\Gamma']}\,|\,\frac{\sigma'}{\gamma'}\,]\!]_{\mathcal{W}}$
$= [\![\, u'[\overrightarrow{e\langle\,\vec{x'}\,\rangle}\,|\,c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma']}]\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}}$

Inductive Case: Distributor $u[\,|\,c\langle\,c\,\rangle\,\overline{[\Gamma]}] \rightsquigarrow_{(\beta)} u'[\,|\,c\langle\,c\,\rangle\,\overline{[\Gamma]}]$
$[\![\, u[\,|\,c\langle\,c\,\rangle\,\overline{[\Gamma]}]\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}} = [\![\, u\overline{[\Gamma]}\,|\,\frac{\sigma-\{c\}\cup\{c\mapsto\bullet\}}{\gamma}\,]\!]_{\mathcal{W}}$
$\overset{\text{I.H.}}{\to}^*_{\mathcal{W}} [\![\, u'[\Gamma]\,|\,\frac{\sigma-\{c\}\cup\{c\mapsto\bullet\}}{\gamma}\,]\!]_{\mathcal{W}} = [\![\, u'[\,|\,c\langle\,c\,\rangle\,\overline{[\Gamma]}]\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}}$

Inductive Case: Distributor $u[\,|\,c\langle\,c\,\rangle\,\overline{[\Gamma]}] \rightsquigarrow_{(\beta)} u[\,|\,c\langle\,c\,\rangle\,\overline{[\Gamma']}]$
$[\![\, u[\,|\,c\langle\,c\,\rangle\,\overline{[\Gamma]}]\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}} = [\![\, u\overline{[\Gamma]}\,|\,\frac{\sigma-\{c\}\cup\{c\mapsto\bullet\}}{\gamma}\,]\!]_{\mathcal{W}}$
$\overset{\text{I.H.}}{\to}^*_{\mathcal{W}} [\![\, u[\Gamma']\,|\,\frac{\sigma-\{c\}\cup\{c\mapsto\bullet\}}{\gamma}\,]\!]_{\mathcal{W}} = [\![\, u[\,|\,c\langle\,c\,\rangle\,\overline{[\Gamma']}]\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}}$

Inductive Case: Distributor $u[\,|\,c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}] \rightsquigarrow_{(\beta)} u'[\,|\,c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}]$
$[\![\, u[\,|\,c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}]\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}} = [\![\, u\overline{[\Gamma]}\,|\,\frac{\sigma'}{\gamma}\,]\!]_{\mathcal{W}}$
$\overset{\text{I.H.}}{\to}^*_{\mathcal{W}} [\![\, u[\Gamma']\,|\,\frac{\sigma'}{\gamma}\,]\!]_{\mathcal{W}} = [\![\, u[\,|\,c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma']}]\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}}$

Inductive Case: Distributor $u[\,|\,c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}] \rightsquigarrow_{(\beta)} u[\,|\,c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma']}]$
$[\![\, u[\,|\,c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}]\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}} = [\![\, u\overline{[\Gamma]}\,|\,\frac{\sigma'}{\gamma}\,]\!]_{\mathcal{W}}$
$\overset{\text{I.H.}}{\to}^*_{\mathcal{W}} [\![\, u[\Gamma']\,|\,\frac{\sigma'}{\gamma}\,]\!]_{\mathcal{W}} = [\![\, u[\,|\,c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma']}]\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}}$ □

Lemma 44. *If $t \rightsquigarrow_{(R,D,L)} u$ and for any $x \in (t)_{bv} \cup (t)_{fp}$, $x \notin (M)_{fv}$ where $\{z \mapsto M\} \subset \sigma$.*

$$[\![\, t\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}} \to^*_{\mathcal{W}} [\![\, u\,|\,\frac{\sigma}{\gamma}\,]\!]_{\mathcal{W}}$$

Proof. We prove this by induction, which is an extension of the proof for Lemma 25. Therefore, we only show the interesting cases.

Case: (r_1)

$$u[\leftarrow s\,t] \rightsquigarrow_R u[\leftarrow s][\leftarrow t]$$

$[\![u[\leftarrow s\,t] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = [\![u \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}[\leftarrow [\![s \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}\, [\![t \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}]$
$\rightarrow_{\mathcal{W}} [\![u \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}[\leftarrow [\![s \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}][\leftarrow [\![t \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}] = [\![u[\leftarrow s][\leftarrow t] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$

Case: (r_2)

$$u[\leftarrow c\langle\, \vec{x}\,\rangle.t] \rightsquigarrow_R u[\,|\, c\langle\, \vec{x}\,\rangle\,[\leftarrow t]]$$

$[\![u[\leftarrow c\langle\, \vec{x}\,\rangle.t] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$

SubCase: $\vec{x} = c$

$[\![u[\leftarrow c\langle\, c\,\rangle.t] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = [\![u \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}[\leftarrow \lambda c.[\![t \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}] \rightarrow_{\mathcal{W}} [\![u \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}[\leftarrow [\![t \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}\{\bullet/c\}]$
$\overset{\text{prop 33}}{=} [\![u[\leftarrow t] \,|\, \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}} = [\![u[\,|\, c\langle\, c\,\rangle\,[\leftarrow t]] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$
where $\sigma' = \sigma \cup \{c \mapsto \bullet\}$

SubCase: $\vec{x} = x_1, \ldots, x_n$

$[\![u[\leftarrow c\langle\, x_1, \ldots, x_n\,\rangle.t] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = [\![u \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}[\leftarrow [\![c\langle\, x_1, \ldots, x_n\,\rangle.t \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}]$
$[\![u \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}[\leftarrow \lambda c.[\![t \,|\, \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}}] \rightarrow_{\mathcal{W}} [\![u \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}[\leftarrow [\![t \,|\, \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}}\{\bullet/c\}]$
$\overset{\text{prop 33}}{=} [\![u[\leftarrow t] \,|\, \frac{\sigma''}{\gamma}]\!]_{\mathcal{W}} = [\![u[\,|\, c\langle\, x_1, \ldots, x_n\,\rangle\,[\leftarrow t]] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$

Case (r_3)

$$u[\,|\, c\langle\, c\,\rangle\,[\leftarrow c]] \rightsquigarrow_R u$$

$[\![u[\,|\, c\langle\, c\,\rangle\,[\leftarrow c]] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = [\![u[\leftarrow c] \,|\, \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}} = [\![u \,|\, \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}}[\leftarrow \bullet]$
$= [\![u \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}[\leftarrow \bullet] \rightarrow_{\mathcal{W}} [\![u \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$

Case (c_2)

$$u[x \leftarrow t] \rightsquigarrow_C u\{t/x\}$$

$[\![u[x \leftarrow t] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = [\![u \,|\, \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}} \overset{\text{prop 34}}{=} [\![u\{t/x\} \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$
where
$\sigma' = \sigma \cup \{x \mapsto [\![t \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}\}$

For the remaining cases, we only show the cases for $[\![u[\leftarrow t] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = [\![u \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}[\leftarrow [\![t \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}]$. The other cases are similar to those in the proof for Lemma 25.

Case: (l_1)

$$s[\leftarrow t]\, u \rightsquigarrow_L (s\, u)[\leftarrow t]$$

$[\![s[\leftarrow t]\, u \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = [\![s \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}[\leftarrow [\![t \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}]\, [\![u \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} \rightarrow_{\mathcal{W}} ([\![s \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}\, [\![u \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}})[\leftarrow [\![t \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}]$
$[\![(s\, u)[\leftarrow t] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$

The proofs for lifting past application (right) (l_2) and sharing (l_4) follow a similar argument so we choose to omit these cases

Case: (l_3)

$$d\langle\, \vec{x}\,\rangle.u[\leftarrow t] \rightsquigarrow_L (d\langle\, \vec{x}\,\rangle.u)[\leftarrow t] \text{ iff } \vec{x} \notin (t)_{fv}$$

SubCase: $\vec{x} = d$

$[\![d\langle d\rangle.u[\leftarrow t] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = \lambda d.([\![u \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}[\leftarrow [\![t \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}]) \rightarrow_{\mathcal{W}} \lambda d.[\![u \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}[\leftarrow [\![t \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}]$
$= [\![(d\langle \vec{x}\rangle.u)[\leftarrow t] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$

SubCase: $\vec{x} = x_1, \ldots, x_n$

$[\![d\langle x_1, \ldots, x_n\rangle.u[\leftarrow t] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = \lambda d.([\![u \,|\, \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}}[\leftarrow [\![t \,|\, \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}}])$
$\rightarrow_{\mathcal{W}} \lambda d.[\![u \,|\, \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}}[\leftarrow [\![t \,|\, \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}}] = [\![(d\langle x_1, \ldots, x_n\rangle.u)[\leftarrow t] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$

Case: (l_5) and (l_6)

$$u[e_1\langle \vec{w}_1\rangle \ldots e_n\langle \vec{w}_n\rangle \,|\, c\langle \vec{x}\rangle\, \overline{[\Gamma]}[\leftarrow t]] \rightsquigarrow_L$$
$$u[e_1\langle \vec{w}_1\rangle \ldots e_n\langle \vec{w}_n\rangle \,|\, c\langle \vec{x}\rangle\, \overline{[\Gamma]}][\leftarrow t]$$

iff all $\vec{x} \notin (t)_{fv}$

$[\![u[e_1\langle \vec{w}_1\rangle \ldots e_n\langle \vec{w}_n\rangle \,|\, c\langle \vec{x}\rangle\, \overline{[\Gamma]}[\leftarrow t]] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$

Case: $\vec{x} = c$

$= [\![u\overline{[\Gamma][\leftarrow t]} \,|\, \frac{\sigma}{\gamma'}]\!]_{\mathcal{W}} = [\![u\overline{[\Gamma]} \,|\, \frac{\sigma}{\gamma'}]\!]_{\mathcal{W}}[\leftarrow [\![t \,|\, \frac{\sigma}{\gamma'}]\!]_{\mathcal{W}}]$
$= [\![u[e_1\langle \vec{w}_1\rangle \ldots e_n\langle \vec{w}_n\rangle \,|\, c\langle c\rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}[\leftarrow [\![t \,|\, \frac{\sigma}{\gamma'}]\!]_{\mathcal{W}}]$
$= [\![u[e_1\langle \vec{w}_1\rangle \ldots e_n\langle \vec{w}_n\rangle \,|\, c\langle c\rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}[\leftarrow [\![t \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}]$
$= [\![u[e_1\langle \vec{w}_1\rangle \ldots e_n\langle \vec{w}_n\rangle \,|\, c\langle c\rangle\, \overline{[\Gamma]}][\leftarrow t] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$

Case: $\vec{x} = x_1, \ldots, x_n$

$= [\![u\overline{[\Gamma][\leftarrow t]} \,|\, \frac{\sigma'}{\gamma'}]\!]_{\mathcal{W}} = [\![u\overline{[\Gamma]} \,|\, \frac{\sigma'}{\gamma'}]\!]_{\mathcal{W}}[\leftarrow [\![t \,|\, \frac{\sigma'}{\gamma'}]\!]_{\mathcal{W}}]$
$= [\![u[e_1\langle \vec{w}_1\rangle \ldots e_n\langle \vec{w}_n\rangle \,|\, c\langle x_1, \ldots, x_n\rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}[\leftarrow [\![t \,|\, \frac{\sigma'}{\gamma'}]\!]_{\mathcal{W}}]$
$= [\![u[e_1\langle \vec{w}_1\rangle \ldots e_n\langle \vec{w}_n\rangle \,|\, c\langle x_1, \ldots, x_n\rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}[\leftarrow [\![t \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}]$
$= [\![u[e_1\langle \vec{w}_1\rangle \ldots e_n\langle \vec{w}_n\rangle \,|\, c\langle x_1, \ldots, x_n\rangle\, \overline{[\Gamma]}][\leftarrow t] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$

And now the new inductive cases not covered in the proof for Lemma 25.

Inductive Case: Weakening: $u[\leftarrow t] \rightsquigarrow_{(R,D,L,C)} u[\leftarrow t']$

$[\![u[\leftarrow t] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = [\![u \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}[\leftarrow [\![t \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}] \overset{\text{I.H.}}{\rightarrow}^*_{\mathcal{W}} [\![u \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}[\leftarrow [\![t' \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}] = [\![u[\leftarrow t'] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$

Inductive Case: Weakening: $u[\leftarrow t] \rightsquigarrow_{(R,D,L,C)} u'[\leftarrow t]$

$[\![u[\leftarrow t] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = [\![u \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}[\leftarrow [\![t \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}] \overset{\text{I.H.}}{\rightarrow}^*_{\mathcal{W}} [\![u' \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}[\leftarrow [\![t \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}] = [\![u'[\leftarrow t] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$

Inductive Case: Distributor $u[\,|\, c\langle c\rangle\, \overline{[\Gamma]}] \rightsquigarrow_{(R,D,L,C)} u'[\,|\, c\langle c\rangle\, \overline{[\Gamma]}]$

$[\![u[\,|\, c\langle c\rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = [\![u\overline{[\Gamma]} \,|\, \frac{\sigma - \{c\} \cup \{c \mapsto \bullet\}}{\gamma}]\!]_{\mathcal{W}}$
$\overset{\text{I.H.}}{\rightarrow}^*_{\mathcal{W}} [\![u'[\Gamma] \,|\, \frac{\sigma - \{c\} \cup \{c \mapsto \bullet\}}{\gamma}]\!]_{\mathcal{W}} = [\![u'[\,|\, c\langle c\rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$

Inductive Case: Distributor $u[\,|\, c\langle c\rangle\, \overline{[\Gamma]}] \rightsquigarrow_{(R,D,L,C)} u[\,|\, c\langle c\rangle\, \overline{[\Gamma']}]$

$[\![u[\,|\, c\langle c\rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = [\![u\overline{[\Gamma]} \,|\, \frac{\sigma - \{c\} \cup \{c \mapsto \bullet\}}{\gamma}]\!]_{\mathcal{W}}$

$\overset{\text{I.H.}}{\to}^*_{\mathcal{W}} [\![u[\Gamma'] \,|\, \frac{\sigma - \{c\} \cup \{c \mapsto \bullet\}}{\gamma}]\!]_{\mathcal{W}} = [\![u[\,|\, c\langle\, c \,\rangle\, \overline{[\Gamma']}] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$

Inductive Case: Distributor $u[\,|\, c\langle\, \vec{x} \,\rangle\, \overline{[\Gamma]}] \rightsquigarrow_{(R,D,L,C)} u'[\,|\, c\langle\, \vec{x} \,\rangle\, \overline{[\Gamma]}]$

$[\![u[\,|\, c\langle\, \vec{x} \,\rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = [\![u\overline{[\Gamma]} \,|\, \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}}$

$\overset{\text{I.H.}}{\to}^*_{\mathcal{W}} [\![u[\Gamma'] \,|\, \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}} = [\![u[\,|\, c\langle\, \vec{x} \,\rangle\, \overline{[\Gamma']}] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$

Inductive Case: Distributor $u[\,|\, c\langle\, \vec{x} \,\rangle\, \overline{[\Gamma]}] \rightsquigarrow_{(R,D,L,C)} u[\,|\, c\langle\, \vec{x} \,\rangle\, \overline{[\Gamma']}]$

$[\![u[\,|\, c\langle\, \vec{x} \,\rangle\, \overline{[\Gamma]}] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}} = [\![u\overline{[\Gamma]} \,|\, \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}}$

$\overset{\text{I.H.}}{\to}^*_{\mathcal{W}} [\![u[\Gamma'] \,|\, \frac{\sigma'}{\gamma}]\!]_{\mathcal{W}} = [\![u[\,|\, c\langle\, \vec{x} \,\rangle\, \overline{[\Gamma']}] \,|\, \frac{\sigma}{\gamma}]\!]_{\mathcal{W}}$ □

Chapter 5

Strong Normalisation of Sharing Reductions

靡不有初
míbùyǒuchū
Everything has a beginning but not always an ending

In this section we formally relate the atomic λ-calculus and the weakening calculus. We use the relationship to show strong normalisation of sharing reductions i.e. $(\rightsquigarrow^*_{(R,D,L,C)})$. This is an important step in proving preservation of strong normalisation for Λ^S_a, as if there is an infinite reduction sequence then it has an infinite β-reduction sequence. This means that (β) is solely responsible for the divergence of reductions.

To show $\rightsquigarrow_{(R,D,L,C)}$ is strongly normalising, we construct a measure that strictly decreases after each reduction step. This measure reasons with *multisets* that intuitively represent (i) the number of copies of shared constructors ($@$ and λ) and (ii) the location of the closure in the term. We show that duplication $(\rightsquigarrow_{(D,R)})$ strictly decreases the number of copies of shared constructors, since duplicating them instantiates the copies, compound of sharings $(\rightsquigarrow_C)$ does not affect the number of copies of shared constructors and strictly decreases the number of closures, and that $(\rightsquigarrow_L)$ strictly decreases the number of steps taken when traversing a term to reach closures.

This is similar to the apporach used in [He18], except He relates her calculus to the λ-calculus rather than the weakening calculus, and uses the reduction rules of the weakening calculus for the cases that involve weakenings.

5.1 Multisets

We prove strong normalisation of sharing reductions through the use of *multisets*. Intuitively, a multiset can be interpreted as a set where elements can be repeated, or equivalently as lists that are considered equal up to the permutation of elements. We use multisets to measure aspects of a term, and show that these aspects strictly decrease via $\rightsquigarrow_{(R,D,L)}$ reduction.

Definition 45 (Multisets)**.** *A* multiset m *is a pair* (A, f) *where* A *is a set and* $f : A \to \mathcal{N}$ *is a function that maps elements of* A *to a natural number.*

The formal definition of multisets in Definition 45 follows intuition when we consider the function f to tell us the number of occurrences of an element $x \in A$ in the multiset m.

Example 46. *Let* $m = (\{x, y, z\}, f)$ *and* $f(x) = 2$, $f(y) = 1$ *and* $f(z) = 3$. *Then this multiset can also be written as* $\{x, x, y, z, z, z\}$ *or equivalently as* $\{x^2, y^1, z^3\}$

Remark : The empty multiset is written as $\{\}$

We will need to be able to reason about multisets in order to use them as part of our reasoning for strong normalisation. First we discuss the union of multisets, which will be needed when measuring a term recursively, e.g. in an application $s\,t$ we will need to measure aspects of s and unionise them with the multiset corresponding to the measure of the same of t, to obtain the overall measure of the application. We follow the same definition provided by Jouannaud and Lescanne in [JL82].

Definition 47 (Union of Multisets)**.** *The* union *(or* sum*) of two multisets* $m = (A, f)$ *and* $n = (B, g)$ *is the multiset* $m \uplus n = (A \cup B, h)$ *such that for all* $x \in A \cup B$, $h(x) = f(x) + g(x)$.

Example 48. *Let* $m = \{a^1, b^3, c^2\}$ *and* $n = \{c^3, d^1\}$, *then* $m \uplus n = \{a^1, b^3, c^5, d^1\}$

Remark : The notion $A \cup B$ is the union of the sets and *not* a disjoint union.

To show strong normalisation of sharing reductions, we need to show that aspects of terms that can be represented as multisets strictly decrease during reduction. In order to show this, we need to be able determine when a multiset is larger/smaller than another i.e. we need to be able to apply an ordering. We follow the definition provided by Huet and Oppen in [HO80].

Definition 49 (Ordering of Multisets)**.** *Given a totally ordered set* A *and two multisets* $m = (A, f)$ *and* $n = (A, g)$, *we say* m *is strictly larger than* n, $m > n$, *if the following conditions hold*

- $m \neq n$
- $\forall x \in A.(g(x) > f(x) \to \exists y \in A.[(y > x) \wedge (f(y) > g(y))])$

Example 50. $\{1^5, 2^2, 3^1\} < \{1^3, 2^4, 3^3\}$

5.2 Sharing Measure

The sharing measure will be the aspect of the term that we show strictly decreases.

5.2.1 Height

The *height* of a term is intuitively a multiset of integers that record the scope of each sharing. The scope is measured by the number of constructors from the sharing node to the root of the term in its graphical notation. The height is defined on terms as $\mathcal{H}^i(-)$, where i is an integer.

Definition 51 (Sharing Height)**.** *The sharing height $\mathcal{H}^i(t)$ of a term t is a multiset given by*

$$\begin{aligned}
\mathcal{H}^i(x) &= \{\} \\
\mathcal{H}^i(s\,t) &= \mathcal{H}^{i+1}(s) \uplus \mathcal{H}^{i+1}(t) \\
\mathcal{H}^i(c\langle\,\vec{x}\,\rangle.t) &= \mathcal{H}^{i+1}(t) \\
\mathcal{H}^i(t[\Gamma]) &= \mathcal{H}^i(t) \uplus \mathcal{H}^i([\Gamma]) \uplus \{i^1\} \\
\mathcal{H}^i([x_1,\ldots,x_n \leftarrow t]) &= \mathcal{H}^{i+1}(t) \\
\mathcal{H}^i([\overrightarrow{e\langle\,\vec{w}\,\rangle}\,|\,c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}]) &= \mathcal{H}^{i+1}(\overline{[\Gamma]}) \uplus \{(i+1)^n\} \\
&\qquad \textit{where } n \textit{ is the number of closures in } \overline{[\Gamma]}
\end{aligned}$$

Notation 52. *For the environment $\overline{[\Gamma]} = [\Gamma_1],\ldots,[\Gamma_n]$, we denote $\mathcal{H}^i([\Gamma_1]) \uplus \cdots \uplus \mathcal{H}^i([\Gamma_n])$ as $\mathcal{H}^i(\overline{[\Gamma]})$*

Notation 53. *We say $\mathcal{H}(t)$ for $\mathcal{H}^1(t)$*

Lemma 54. *If $t \rightsquigarrow_{(L)} u$ then $\mathcal{H}^i(t) > \mathcal{H}^i(u)$*

Proof. We prove this on a case-by-case basis

$$s[\Gamma]\,t \rightsquigarrow_L (s\,t)[\Gamma]$$

$\mathcal{H}^i((s[\Gamma])\,t) = \mathcal{H}^{i+1}(s[\Gamma]) \uplus \mathcal{H}^{i+1}(t) = \mathcal{H}^{i+1}(s) \uplus \mathcal{H}^{i+1}(t) \uplus \mathcal{H}^{i+1}([\Gamma]) \uplus \{i+1\}$
$\mathcal{H}^i((s\,t)[\Gamma]) = \mathcal{H}^i(s\,t) \uplus \mathcal{H}^i([\Gamma]) = \mathcal{H}^{i+1}(s) \uplus \mathcal{H}^{i+1}(t) \uplus \mathcal{H}^i([\Gamma]) \uplus \{i\}$

$$s\,t[\Gamma] \rightsquigarrow_L (s\,t)[\Gamma]$$

This case is similar to the one above and we omit it.

$$d\langle\,\vec{x}\,\rangle.t[\Gamma] \rightsquigarrow_L (d\langle\,\vec{x}\,\rangle.t)[\Gamma] \text{ iff all } \vec{x} \in (t)_{fv}$$

$\mathcal{H}^i(c\langle\,\vec{x}\,\rangle.t[\Gamma]) = \mathcal{H}^{i+1}(t[\Gamma]) = \mathcal{H}^{i+1}(t) \uplus \mathcal{H}^{i+1}([\Gamma]) \uplus \{i+1\}$
$\mathcal{H}^i((c\langle\,\vec{x}\,\rangle.t)[\Gamma]) = \mathcal{H}^i(c\langle\,\vec{x}\,\rangle.t) \uplus \mathcal{H}^i([\Gamma]) \uplus \{i\} = \mathcal{H}^{i+1}(t) \uplus \mathcal{H}^i([\Gamma]) \uplus \{i\}$

$$u[\vec{x} \leftarrow t[\Gamma]] \rightsquigarrow_L u[\vec{x} \leftarrow t][\Gamma]$$

$\mathcal{H}^i(u[\vec{x} \leftarrow t[\Gamma]]) = \mathcal{H}^i(u) \uplus \mathcal{H}^i([\vec{x} \leftarrow t[\Gamma]]) \uplus \{i\} = \mathcal{H}^i(u) \uplus \mathcal{H}^{i+1}(t) \uplus \mathcal{H}^{i+1}([\Gamma]) \uplus \{i, i+1\}$
$\mathcal{H}^i(u[\vec{x} \leftarrow t][\Gamma]) = \mathcal{H}^i(u[\vec{x} \leftarrow t]) \uplus \mathcal{H}^i([\Gamma]) \uplus \{i\} = \mathcal{H}^i(u) \uplus \mathcal{H}^{i+1}(t) \uplus \mathcal{H}^{i+1}([\Gamma]) \uplus \{i, i\}$

$$u[e_1\langle\,\vec{w}_1\,\rangle \ldots e_n\langle\,\vec{w}_n\,\rangle\,|\,c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}[\vec{y} \leftarrow t]] \rightsquigarrow_L$$

$$u\{(\vec{w}_1/\vec{y})/e_1\}_b \dots \{(\vec{w}_n/\vec{y})/e_n\}_b[e_1\langle\,\vec{w}_1/\vec{y}\,\rangle \dots e_n\langle\,\vec{w}_n/\vec{y}\,\rangle \,|\, c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}][\vec{y} \leftarrow t]$$

iff all $\vec{x} \notin (t)_{fv}$
$\mathcal{H}^i(u[e_1\langle\,\vec{w}_1\,\rangle \dots e_n\langle\,\vec{w}_n\,\rangle \,|\, c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}[\vec{y} \leftarrow t]])$
$= \mathcal{H}^i(u) \uplus \mathcal{H}^i([e_1\langle\,\vec{w}_1\,\rangle \dots e_n\langle\,\vec{w}_n\,\rangle \,|\, c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}[\vec{y} \leftarrow t]]) \uplus \{i\}$
$= \mathcal{H}^i(u) \uplus \mathcal{H}^{i+1}(\overline{[\Gamma]}) \uplus \mathcal{H}^{i+1}([\vec{y} \leftarrow t]) \uplus \{i, (i+1)^{n+1}\}$
where n is the number of closures in the environment $\overline{[\Gamma]}$
$= \mathcal{H}^i(u) \uplus \mathcal{H}^{i+1}(\overline{[\Gamma]}) \uplus \mathcal{H}^{i+2}(t) \uplus \{i, (i+1)^{n+1}\}$
$\mathcal{H}^i(u\{(\vec{w}_1/\vec{y})/e_1\}_b \dots \{(\vec{w}_n/\vec{y})/e_n\}_b[e_1\langle\,\vec{w}_1/\vec{y}\,\rangle \dots e_n\langle\,\vec{w}_n/\vec{y}\,\rangle \,|\, c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}][\vec{y} \leftarrow t])$
$= \mathcal{H}^i(u\{(\vec{w}_1/\vec{y})/e_1\}_b \dots \{(\vec{w}_n/\vec{y})/e_n\}_b[e_1\langle\,\vec{w}_1/\vec{y}\,\rangle \dots e_n\langle\,\vec{w}_n/\vec{y}\,\rangle \,|\, c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}]) \uplus \mathcal{H}^{i+1}(t) \uplus \{i\}$
$= \mathcal{H}^i(u\{(\vec{w}_1/\vec{y})/e_1\}_b \dots \{(\vec{w}_n/\vec{y})/e_n\}_b) \uplus \mathcal{H}^{i+1}(\overline{[\Gamma]}) \uplus \mathcal{H}^{i+1}(t) \uplus \{i^2, (i+1)^n\}$
$= \mathcal{H}^i(u) \uplus \mathcal{H}^{i+1}(\overline{[\Gamma]}) \uplus \mathcal{H}^{i+1}(t) \uplus \{i^2, (i+1)^n\}$

$$u[e_1\langle\,\vec{w}_1\,\rangle \dots e_n\langle\,\vec{w}_n\,\rangle \,|\, c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}[\overrightarrow{f\langle\,\vec{z}\,\rangle} \,|\, d\langle\,\vec{a}\,\rangle\,\overline{[\Gamma']}]] \rightsquigarrow_L$$

$$u\{(\vec{w}_1/\vec{z})/e_1\}_b \dots \{(\vec{w}_n/\vec{z})/e_n\}_b[e_1\langle\,\vec{w}_1/\vec{z}\,\rangle \dots e_n\langle\,\vec{w}_n/\vec{z}\,\rangle \,|\, c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}][\overrightarrow{f\langle\,\vec{z}\,\rangle} \,|\, d\langle\,\vec{a}\,\rangle\,\overline{[\Gamma']}]$$

iff all $\vec{x} \in (u[e_1\langle\,\vec{w}_1\,\rangle \dots e_n\langle\,\vec{w}_n\,\rangle \,|\, c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}])_{fv}$
$\mathcal{H}^i(u[e_1\langle\,\vec{w}_1\,\rangle \dots e_n\langle\,\vec{w}_n\,\rangle \,|\, c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}[\overrightarrow{f\langle\,\vec{z}\,\rangle} \,|\, d\langle\,\vec{a}\,\rangle\,\overline{[\Gamma']}]])$
$= \mathcal{H}^i(u) \uplus \mathcal{H}^{i+1}(\overline{[\Gamma]}) \uplus \mathcal{H}^{i+1}([\overrightarrow{f\langle\,\vec{z}\,\rangle} \,|\, d\langle\,\vec{a}\,\rangle\,\overline{[\Gamma']}]) \uplus \{i, (i+1)^{n+1}\}$
where n is the number of closures in $\overline{[\Gamma]}$
$= \mathcal{H}^i(u) \uplus \mathcal{H}^{i+1}(\overline{[\Gamma]}) \uplus \mathcal{H}^{i+2}(\overline{[\Gamma']}) \uplus \{i, (i+1)^{n+1}, (i+2)^m\}$
where m is the number of closures in $\overline{[\Gamma']}$
$\mathcal{H}^i(u\{(\vec{w}_1/\vec{z})/e_1\}_b \dots \{(\vec{w}_n/\vec{z})/e_n\}_b[e_1\langle\,\vec{w}_1/\vec{z}\,\rangle \dots e_n\langle\,\vec{w}_n/\vec{z}\,\rangle \,|\, c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}][\overrightarrow{f\langle\,\vec{z}\,\rangle} \,|\, d\langle\,\vec{a}\,\rangle\,\overline{[\Gamma']}])$
$\mathcal{H}^i(u\{(\vec{w}_1/\vec{z})/e_1\}_b \dots \{(\vec{w}_n/\vec{z})/e_n\}_b[e_1\langle\,\vec{w}_1/\vec{z}\,\rangle \dots e_n\langle\,\vec{w}_n/\vec{z}\,\rangle \,|\, c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}])$
$\quad \uplus \mathcal{H}^{i+1}(\overline{[\Gamma']}) \uplus \{i, (i+1)^m\}$
$= \mathcal{H}^i(u\{(\vec{w}_1/\vec{z})/e_1\}_b \dots \{(\vec{w}_n/\vec{z})/e_n\}_b) \uplus \mathcal{H}^{i+1}(\overline{[\Gamma]}) \uplus \mathcal{H}^{i+1}(\overline{[\Gamma']}) \uplus \{i, (i+1)^{n+m}\}$
$= \mathcal{H}^i(u) \uplus \mathcal{H}^{i+1}(\overline{[\Gamma]}) \uplus \mathcal{H}^{i+1}(\overline{[\Gamma']}) \uplus \{i, (i+1)^{n+m}\}$ □

5.2.2 Weight

The *weight* of a term is intuitively the number or copies each constructor (abstraction, application and variable) will exist after duplication. Figure 5.1 illustrates this, by showing a side-by-side comparison of the term

$$x\langle\,x\,\rangle.c_1\langle\,w_1\,\rangle.w_1\,((c_2\langle\,w_2\,\rangle.w_2)\,x)$$

$$[c_1\langle\,w_1\,\rangle c_2\langle\,w_2\,\rangle \,|\, y\langle\,y\,\rangle\,[w_1, w_2 \leftarrow z\langle\,z\,\rangle.z_1\,(z_2\,y)[z_1, z_2 \leftarrow z]]]$$

and its equivalent in the $\Lambda_{\mathcal{W}}$-calculus obtained by $[\![-]\!]^{\mathcal{W}}$. Each red line shows the connection between the abstraction and application constructors in both calculi. The weight of a constructor is then the number of red lines associated with it, e.g. the weight of the example is the multiset $\{1^6, 2^4, 4^1\}$.

Calculuating the weight of a term requires an auxiliary function from variables to integers. This function is defined by assigning integer weights to the variables of a term. This auxiliary function is defined on terms $\mathcal{V}^i(-)$, where i is an integer. To measure variables

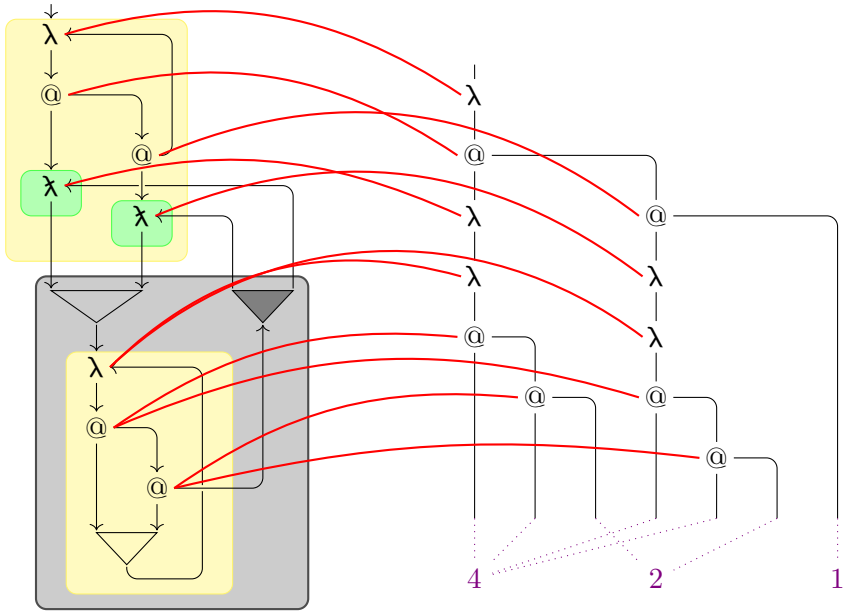


Figure 5.1: The weight is the multiset of incoming red arcs for each application and abstraction; here $\{1^5, 2^3\}$, together with the number of purple dotted lines for each variable; here $\{1, 2, 4\}$. Thus the overall weight is $\{1^6, 2^4, 4\}$

independently of binders is vital. It allows to measure distributors, which duplicate λ's but not the bound variable. Also, only bound variables for abstractions are measured since variables bound by sharings are substituted in the interpretation.

Definition 55 (Variable weights)**.** *The function $\mathcal{V}^i(t)$ returns a function that assigns integer weights to the free variables of t. It is defined by the following*

$$\begin{aligned}
\mathcal{V}^i(x) &= \{x \mapsto i\} \\
\mathcal{V}^i(s\,t) &= \mathcal{V}^i(s) \uplus \mathcal{V}^i(t) \\
\mathcal{V}^i(c\langle\, c\,\rangle.t) &= \mathcal{V}^i(t)/\{c\} \\
\mathcal{V}^i(c\langle\, \vec{x}\,\rangle.t) &= \mathcal{V}^i(t) \uplus \{c \mapsto i\} \\
\mathcal{V}^i(t[\leftarrow s]) &= \mathcal{V}^i(t) \uplus \mathcal{V}^1(s) \\
\mathcal{V}^i(t[x_1, \ldots, x_n \leftarrow s]) &= \mathcal{V}^i(t)/\{x_1, \ldots, x_n\} \uplus \mathcal{V}^{f(x_1)+\cdots+f(x_n)}(s) \\
&\quad \textit{where } f = \mathcal{V}^i(t) \\
\mathcal{V}^i(t[e_1\langle\, \vec{w}_1\,\rangle \ldots e_n\langle\, \vec{w}_n\,\rangle \,|\, c\langle\, c\,\rangle\, \overline{[\Gamma]}]) &= \mathcal{V}^i(t\overline{[\Gamma]})/\{c, e_1, \ldots, e_n\} \\
\mathcal{V}^i(t[e_1\langle\, \vec{w}_1\,\rangle \ldots e_n\langle\, \vec{w}_n\,\rangle \,|\, c\langle\, \vec{x}\,\rangle\, \overline{[\Gamma]}]) &= \mathcal{V}^i(t\overline{[\Gamma]})/\{e_1, \ldots, e_n\} \uplus \{c \mapsto i\}
\end{aligned}$$

The weight of a term can then be defined via the use of this auxiliary function. The auxiliary function is used when calculating the weight of a sharing, where the sharing weight of the variables bound by the sharing play a significant role in calculuating the weight of the shared term. In the case of a weakening, we assign an initial weight of 1 to indicate that the constructor is not duplicated by appears at least once in the weakening calculus.

Definition 56 (Sharing Weight)**.** *The sharing weight $\mathcal{W}^i(t)$ of a term t is a multiset of integers computed by the function defined below*

$$\begin{aligned}
\mathcal{W}^i(x) &= \{\} \\
\mathcal{W}^i(s\,t) &= \mathcal{W}^i(s) \uplus \mathcal{W}^i(t) \uplus \{i\} \\
\mathcal{W}^i(c\langle\, c\,\rangle.t) &= \mathcal{W}^i(t) \uplus \{i\} \uplus \{\mathcal{V}^i(t)(c)\} \\
\mathcal{W}^i(c\langle\, \vec{x}\,\rangle.t) &= \mathcal{W}^i(t) \uplus \{i\} \\
\mathcal{W}^i(t[\leftarrow s]) &= \mathcal{W}^i(t) \uplus \mathcal{W}^1(s) \\
\mathcal{W}^i(t[x_1,\ldots,x_n \leftarrow s]) &= \mathcal{W}^i(t) \uplus \mathcal{W}^{f(x_1)+\cdots+f(x_n)}(s) \\
&\quad \textit{where } f = \mathcal{V}^i(t) \\
\mathcal{W}^i(t[e_1\langle\, \vec{w}_1\,\rangle \ldots e_n\langle\, \vec{w}_n\,\rangle \,|\, c\langle\, c\,\rangle\, \overline{[\Gamma]}]) &= \mathcal{W}^i(t\overline{[\Gamma]}) \uplus \{\mathcal{V}^i(t\overline{[\Gamma]})(c)\} \\
\mathcal{W}^i(t[e_1\langle\, \vec{w}_1\,\rangle \ldots e_n\langle\, \vec{w}_n\,\rangle \,|\, c\langle\, \vec{x}\,\rangle\, \overline{[\Gamma]}]) &= \mathcal{W}^i(t\overline{[\Gamma]})
\end{aligned}$$

Notation 57. *We say $\mathcal{W}(t)$ for $\mathcal{W}^1(t)$*

Proposition 58. *For $e \notin \vec{w}$, $\mathcal{W}^i(t) = \mathcal{W}^i(t\{\vec{w}/e\}_b)$*

Proof. To prove this, first we need to prove that book-keeping does not affect the function $\mathcal{V}^i(t)$. We prove this by induction on t.
Base Case: Variable
Vacuously True

Base Case: Abstraction
$\mathcal{V}^i(e\langle\, \vec{y}\,\rangle.t\{\vec{w}/e\}_b) = \mathcal{V}^i(e\langle\, \vec{w}\,\rangle.t) = \mathcal{V}^i(t) \uplus \{e \mapsto i\} = \mathcal{V}^i(e\langle\, \vec{y}\,\rangle.t)$

Base Case: Distributor
$\mathcal{V}^i(u[\overrightarrow{f\langle\, \vec{z}\,\rangle} \,|\, e\langle\, \vec{y}\,\rangle\, \overline{[\Gamma]}]\{\vec{w}/e\}_b) = \mathcal{V}^i(u[\overrightarrow{f\langle\, \vec{z}\,\rangle} \,|\, e\langle\, \vec{w}\,\rangle\, \overline{[\Gamma]}])$
$= \mathcal{V}^i(u\overline{[\Gamma]})\ \{\vec{e}\} = \mathcal{V}^i(u[\overrightarrow{f\langle\, \vec{z}\,\rangle} \,|\, e\langle\, \vec{y}\,\rangle\, \overline{[\Gamma]}])$

Inductive Case: Application
$\mathcal{V}^i(s\,t\{\vec{w}/e\}_b) = \mathcal{V}^i((s\{\vec{w}/e\}_b)\,t\{\vec{w}/e\}_b) = \mathcal{V}^i(s\{\vec{w}/e\}_b) \uplus \mathcal{V}^i(t\{\vec{w}/e\}_b) \overset{\text{I.H.}^2}{=} \mathcal{V}^i(s) \uplus \mathcal{V}^i(t) = \mathcal{V}^i(s\,t)$

Inductive Case: Abstraction
Case 1
$\mathcal{V}^i((c\langle\, c\,\rangle.t)\{\vec{w}/e\}_b) = \mathcal{V}^i(c\langle\, c\,\rangle.t\{\vec{w}/e\}_b) = \mathcal{V}^i(t\{\vec{w}/e\}_b)/\{c\} \overset{\text{I.H.}}{=} \mathcal{V}^i(t)/\{c\} = \mathcal{V}^i(c\langle\, c\,\rangle.t)$
Case 2
$\mathcal{V}^i((c\langle\, \vec{x}\,\rangle.t)\{\vec{w}/e\}_b) = \mathcal{V}^i(c\langle\, \vec{x}\,\rangle.t\{\vec{w}/e\}_b) = \mathcal{V}^i(t\{\vec{w}/e\}_b) \uplus \{c \mapsto i\} \overset{\text{I.H.}}{=} \mathcal{V}^i(t) \uplus \{c \mapsto i\} = \mathcal{V}^i(c\langle\, \vec{x}\,\rangle.t)$

Inductive Case: Weakening
$\mathcal{V}^i(u[\leftarrow t]\{\vec{w}/e\}_b) = \mathcal{V}^i(u\{\vec{w}/e\}_b[\leftarrow t\{\vec{w}/e\}_b]) = \mathcal{V}^i(u\{\vec{w}/e\}_b) \uplus \mathcal{V}^1(t\{\vec{w}/e\}_b)$

$\overset{\text{I.H.}^2}{=} \mathcal{V}^i(u) \uplus \mathcal{V}^1(t) = \mathcal{V}^i(u[\leftarrow t])$

Inductive Case: Sharing
$\mathcal{V}^i(u[x_1 \dots x_n \leftarrow t]\{\vec{w}/e\}_b) = \mathcal{V}^i(u\{\vec{w}/e\}_b[x_1 \dots x_n \leftarrow t\{\vec{w}/e\}_b])$
$= (\mathcal{V}^i(u\{\vec{w}/e\}_b)/\{x_1, \dots, x_n\}) \uplus \mathcal{V}(tj\{\vec{w}/e\}_b)$ where $j = \mathcal{V}^i(t\{\vec{w}/e\}_b) + \dots + \mathcal{V}^i(t\{\vec{w}/e\}_b)$
$\overset{\text{I.H.}^{n+2}}{=} (\mathcal{V}^i(u)/\{x_1, \dots, x_n\}) \uplus \mathcal{V}(t)$ where $j = \mathcal{V}^i(t) + \dots + \mathcal{V}^i(t) = \mathcal{V}^i(u[x_1, \dots, x_n \leftarrow t])$

Inductive Case: Distributor
Case 1
$\mathcal{V}^i(u[\overrightarrow{f\langle\, \vec{z}\,\rangle} \mid c\langle\, c\,\rangle\, \overline{[\Gamma]}]\{\vec{w}/e\}_b) = \mathcal{V}^i(u[\overrightarrow{f\langle\, \vec{z}\,\rangle} \mid c\langle\, c\,\rangle\, \overline{[\Gamma]}\{\vec{w}/e\}_b]) = \mathcal{V}^i(u\overline{[\Gamma]}\{\vec{w}/e\}_b)/\{c, \vec{f}\}$
$\overset{\text{I.H.}}{=} \mathcal{V}^i(u\overline{[\Gamma]})/\{c, \vec{f}\} = \mathcal{V}^i(u[\overrightarrow{f\langle\, \vec{z}\,\rangle} \mid c\langle\, c\,\rangle\, \overline{[\Gamma]}])$
Case 2
$\mathcal{V}^i(u[\overrightarrow{f\langle\, \vec{z}\,\rangle} \mid c\langle\, \vec{x}\,\rangle\, \overline{[\Gamma]}]\{\vec{w}/e\}_b) = \mathcal{V}^i(u[\overrightarrow{f\langle\, \vec{z}\,\rangle} \mid c\langle\, \vec{x}\,\rangle\, \overline{[\Gamma]}\{\vec{w}/e\}_b])$
$= \mathcal{V}^i(u\overline{[\Gamma]}\{\vec{w}/e\}_b)/\{\vec{f}\} \uplus \{c \mapsto i\}$
$\overset{\text{I.H.}}{=} \mathcal{V}^i(u\overline{[\Gamma]})/\{\vec{f}\} \uplus \{c \mapsto i\} = \mathcal{V}^i(u[\overrightarrow{f\langle\, \vec{z}\,\rangle} \mid c\langle\, \vec{x}\,\rangle\, \overline{[\Gamma]}])$

We now prove this proposition by induction on t
Base Case: Variable
$\mathcal{W}^i(x\{\vec{w}/e\}_b) = \mathcal{W}^i(x)$

Base Case: Abstraction
$\mathcal{W}^i(e\langle\, \vec{y}\,\rangle.t\{\vec{w}/e\}_b) = \mathcal{W}^i(e\langle\, \vec{w}\,\rangle.t) = \mathcal{W}^i(t) \uplus \{i\} = \mathcal{W}^i(e\langle\, \vec{y}\,\rangle.t)$

Base Case: Distributor
$\mathcal{W}^i(u[\overrightarrow{e\langle\, \vec{z}\,\rangle} \mid e\langle\, \vec{y}\,\rangle\, \overline{[\Gamma]}]\{\vec{w}/e\}_b) = \mathcal{W}^i(u[\overrightarrow{e\langle\, \vec{z}\,\rangle} \mid e\langle\, \vec{w}\,\rangle\, \overline{[\Gamma]}]) = \mathcal{W}^i(u\overline{[\Gamma]})$
$= \mathcal{W}^i(u[\overrightarrow{e\langle\, \vec{z}\,\rangle} \mid e\langle\, \vec{y}\,\rangle\, \overline{[\Gamma]}])$

Inductive Case: Application
$\mathcal{W}^i(s\,t\{\vec{w}/e\}_b) = \mathcal{W}^i((s\{\vec{w}/e\}_b)\,t\{\vec{w}/e\}_b) = \mathcal{W}^i(s\{\vec{w}/e\}_b) \uplus \mathcal{W}^i(t\{\vec{w}/e\}_b) \uplus \{i\}$
$\overset{\text{I.H.}^2}{=} \mathcal{W}^i(s) \uplus \mathcal{W}^i(t) \uplus \{i\} = \mathcal{W}^i(s\,t)$

Inductive Case: Abstraction
Case 1
$\mathcal{W}^i((c\langle\, c\,\rangle.t)\{\vec{w}/e\}_b) = \mathcal{W}^i(c\langle\, c\,\rangle.t\{\vec{w}/e\}_b) = \mathcal{W}^i(t\{\vec{w}/e\}_b) \uplus \{i, \mathcal{V}^i(t\{\vec{w}/e\}_b)(c)\}$
$\overset{\text{I.H.}}{=} \mathcal{W}^i(t) \uplus \{i, \mathcal{V}^i(t)(c)\} = \mathcal{W}^i(c\langle\, c\,\rangle.t)$
Case 2
$\mathcal{W}^i((c\langle\, \vec{x}\,\rangle.t)\{\vec{w}/e\}_b) = \mathcal{W}^i(c\langle\, \vec{x}\,\rangle.t\{\vec{w}/e\}_b) = \mathcal{W}^i(t\{\vec{w}/e\}_b) \uplus \{i\} \overset{\text{I.H.}}{=} \mathcal{W}^i(t) \uplus \{i\}$
$= \mathcal{W}^i(c\langle\, \vec{x}\,\rangle.t)$

Inductive Case: Weakening
$\mathcal{W}^i(u[\leftarrow t]\{\vec{w}/e\}_b) = \mathcal{W}^i(u\{\vec{w}/e\}_b[\leftarrow t\{\vec{w}/e\}_b]) = \mathcal{W}^i(u\{\vec{w}/e\}_b) \uplus \mathcal{W}^1(t\{\vec{w}/e\}_b)$
$\overset{\text{I.H.}^2}{=} \mathcal{W}^i(u) \uplus \mathcal{W}^1(t) = \mathcal{W}^i(u[\leftarrow t])$

Inductive Case: Sharing
$\mathcal{W}^i(u[x_1,\ldots,x_n \leftarrow t]\{\vec{w}/e\}_b) = \mathcal{W}^i(u\{\vec{w}/e\}_b[x_1,\ldots,x_n \leftarrow t\{\vec{w}/e\}_b])$
$= \mathcal{W}^i(u\{\vec{w}/e\}_b) \uplus \mathcal{W}^j(t\{\vec{w}/e\}_b)$ where $j = \mathcal{V}^i(u\{\vec{w}/e\}_b)(x_1) + \cdots + \mathcal{V}^i(u\{\vec{w}/e\}_b)(x_n)$
$\overset{\text{I.H.}^{n+2}}{=} \mathcal{W}^i(u) \uplus \mathcal{W}^j(t)$ where $j = \mathcal{V}^i(u)(x_1) + \cdots + \mathcal{V}^i(u)(x_1) = \mathcal{W}^i(u[x_1,\ldots,x_n \leftarrow t])$

Inductive Case: Distributor
Case 1
$\mathcal{W}^i(u[\overrightarrow{f\langle\,\vec{z}\,\rangle}\,|\,c\langle\,c\,\rangle\,\overline{[\Gamma]}]\{\vec{w}/e\}_b) = \mathcal{W}^i(u[\overrightarrow{f\langle\,\vec{z}\,\rangle}\,|\,c\langle\,c\,\rangle\,\overline{[\Gamma]}\{\vec{w}/e\}_b])$
$= \mathcal{W}^i(u\overline{[\Gamma]}\{\vec{w}/e\}_b) \uplus \{\mathcal{V}^i(u\overline{[\Gamma]}\{\vec{w}/e\}_b)(c)\} \overset{\text{I.H.}}{=} \mathcal{W}^i(u\overline{[\Gamma]}) \uplus \{\mathcal{V}^i(u\overline{[\Gamma]})(c)\}$
$= \mathcal{W}^i(u[\overrightarrow{f\langle\,\vec{z}\,\rangle}\,|\,c\langle\,c\,\rangle\,\overline{[\Gamma]}])$
Case 2
$\mathcal{W}^i(u[\overrightarrow{f\langle\,\vec{z}\,\rangle}\,|\,c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}]\{\vec{w}/e\}_b) = \mathcal{W}^i(u[\overrightarrow{f\langle\,\vec{z}\,\rangle}\,|\,c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}\{\vec{w}/e\}_b])$
$= \mathcal{W}^i(u\overline{[\Gamma]}\{\vec{w}/e\}_b) \overset{\text{I.H.}}{=} \mathcal{W}^i(u\overline{[\Gamma]}) = \mathcal{W}^i(u[\overrightarrow{f\langle\,\vec{z}\,\rangle}\,|\,c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}])$ □

Lemma 59. *If $t \rightsquigarrow_{(R,D)} u$ then $\mathcal{W}^i(t) > \mathcal{W}^i(u)$*

Proof. We prove this on a case-by-case basis

Deletion Rules

$$u[\leftarrow s\,t] \rightsquigarrow_R u[\leftarrow s][\leftarrow t]$$

$\mathcal{W}^i(u[\leftarrow s\,t]) = \mathcal{W}^i(u) \uplus \mathcal{W}^1(s\,t) = \mathcal{W}^i(u) \uplus \mathcal{W}^1(s) \uplus \mathcal{W}^1(t) \uplus \{1\}$
$\mathcal{W}^i(u[\leftarrow s][\leftarrow t]) = \mathcal{W}^i(u[\leftarrow s]) \uplus \mathcal{W}^1(t) = \mathcal{W}^i(u) \uplus \mathcal{W}^1(s) \uplus \mathcal{W}^1(t)$

$$u[\leftarrow c\langle\,\vec{x}\,\rangle.t] \rightsquigarrow_R u[\,|\,c\langle\,\vec{x}\,\rangle\,[\leftarrow t]]$$

Case 1:
$\mathcal{W}^i(u[\leftarrow c\langle\,c\,\rangle.t]) = \mathcal{W}^i(u) \uplus \mathcal{W}^1(c\langle\,c\,\rangle.t) = \mathcal{W}^i(u) \uplus \mathcal{W}^1(t) \uplus \{1, \mathcal{V}^1(t)(c)\}$
$\mathcal{W}^i(u[\,|\,c\langle\,c\,\rangle\,[\leftarrow t]]) = \mathcal{W}^i(u[\leftarrow t]) \uplus \{\mathcal{V}^i(u[\leftarrow t])(c)\}$
$= \mathcal{W}^i(u) \uplus \mathcal{W}^1(t) \uplus \{(\mathcal{V}^i(u) \uplus \mathcal{V}^1(t))(c)\} = \mathcal{W}^i(u) \uplus \mathcal{W}^1(t) \uplus \{\mathcal{V}^1(t)(c)\}$
Case 2:
$\mathcal{W}^i(u[\leftarrow c\langle\,\vec{x}\,\rangle.t]) = \mathcal{W}^i(u) \uplus \mathcal{W}^1(c\langle\,\vec{x}\,\rangle.t) = \mathcal{W}^i(u) \uplus \mathcal{W}^1(t) \uplus \{1\}$
$\mathcal{W}^i(u[\,|\,c\langle\,\vec{x}\,\rangle\,[\leftarrow t]]) = \mathcal{W}^i(u[\leftarrow t]) = \mathcal{W}^i(u) \uplus \mathcal{W}^1(t)(c)$

$$u[\,|\,c\langle\,c\,\rangle\,[\leftarrow c]] \rightsquigarrow_R u$$

$\mathcal{W}^i(u[\,|\,c\langle\,c\,\rangle\,[\leftarrow c]]) = \mathcal{W}^i(u[\leftarrow c]) \uplus \{\mathcal{V}^i(u[\leftarrow c])(c)\}$
$= \mathcal{W}^i(u) \uplus \mathcal{W}^1(c) \uplus \{1\} = \mathcal{W}^i(u) \uplus \{1\}$

Duplication Rules

$$u^*[x_1 \ldots x_n \leftarrow s\,t] \rightsquigarrow_D u^*\{z_1\,y_1/x_1\}\ldots\{z_n\,y_n/x_n\}[z_1 \ldots z_n \leftarrow s][y_1 \ldots y_n \leftarrow t]$$

$\mathcal{W}^i(u^*[x_1 \ldots x_n \leftarrow s\,t]) = \mathcal{W}^i(u) \uplus \mathcal{W}^j(s\,t) = \mathcal{W}^i(u) \uplus \mathcal{W}^j(s) \uplus \mathcal{W}^j(s) \uplus \{j\}$
where $j = \mathcal{V}^i(u)(x_1) + \cdots + \mathcal{V}^i(u)(x_n)$
$\mathcal{W}^i(u^*\{z_1\,y_1/x_1\}\ldots\{z_n\,y_n/x_n\}[z_1 \ldots z_n \leftarrow s][y_1 \ldots y_n \leftarrow t])$

$= \mathcal{W}^i(u^*\{z_1\, y_1/x_1\}\ldots\{z_n\, y_n/x_n\}[z_1\ldots z_n \leftarrow s]) \uplus \mathcal{W}^k(t)$
$= \mathcal{W}^i(u^*\{z_1\, y_1/x_1\}\ldots\{z_n\, y_n/x_n\}) \uplus \mathcal{W}^l(s) \uplus \mathcal{W}^k(t)$
where $k = \mathcal{V}^i(u^*\{z_1\, y_1/x_1\}\ldots\{z_n\, y_n/x_n\}[z_1\ldots z_n \leftarrow s])(y_1) + \ldots$
$\quad \cdots + \mathcal{V}^i(u^*\{z_1\, y_1/x_1\}\ldots\{z_n\, y_n/x_n\}[z_1\ldots z_n \leftarrow s])(y_n)$
$= \mathcal{V}^i(u^*\{z_1\, y_1/x_1\}\ldots\{z_n\, y_n/x_n\})(y_1) + \cdots + \mathcal{V}^i(u^*\{z_1\, y_1/x_1\}\ldots\{z_n\, y_n/x_n\})(y_n)$
$= \mathcal{V}^i(u)(x_1) + \cdots + \mathcal{V}^i(u)(x_n) = j$
and where $l = \mathcal{V}^i(u^*\{z_1\, y_1/x_1\}\ldots\{z_n\, y_n/x_n\})(z_1) + \ldots$
$\quad \cdots + \mathcal{V}^i(u^*\{z_1\, y_1/x_1\}\ldots\{z_n\, y_n/x_n\})(z_n)$
$= \mathcal{V}^i(u)(x_1) + \cdots + \mathcal{V}^i(u)(x_n) = j$
Therefore
$= \mathcal{W}^i(u^*\{z_1\, y_1/x_1\}\ldots\{z_n\, y_n/x_n\}) \uplus \mathcal{W}^j(s) \uplus \mathcal{W}^j(t)$
$= \mathcal{W}^i(u) \uplus \mathcal{W}^j(s) \uplus \mathcal{W}^j(t) \uplus \{\mathcal{V}^i(u)(x_1), \ldots, \mathcal{V}^i(u)(x_n)\}$

$$u[x_1, \ldots, x_n \leftarrow c\langle\, \vec{y}\,\rangle.t] \rightsquigarrow_D$$
$$u\{e_i\langle\, w_1^i\,\rangle.w_1^i/x_i\}_{1\le i\le n}[e_1\langle\, w_1^1\,\rangle \ldots e_n\langle\, w_1^n\,\rangle \,|\, c\langle\, \vec{y}\,\rangle\,[w_1^1, \ldots, w_1^n \leftarrow t]]$$

Case 1:
$\mathcal{W}^i(u[x_1, \ldots, x_n \leftarrow c\langle\, c\,\rangle.t]) = \mathcal{W}^i(u) \uplus \mathcal{W}^j(c\langle\, c\,\rangle.t) = \mathcal{W}^i(u) \uplus \mathcal{W}^j(t) \uplus \{j, \mathcal{V}^j(t)(c)\}$
where $j = \mathcal{V}^i(u)(x_1) + \cdots + \mathcal{V}^i(u)(x_n)$
$\mathcal{W}^i(u\{e_i\langle\, w_1^i\,\rangle.w_1^i/x_i\}_{1\le i\le n}[e_1\langle\, w_1^1\,\rangle \ldots e_n\langle\, w_1^n\,\rangle \,|\, c\langle\, c\,\rangle\,[w_1^1, \ldots, w_1^n \leftarrow t]])$
$= \mathcal{W}^i(u\{e_i\langle\, w_1^i\,\rangle.w_1^i/x_i\}_{1\le i\le n}[w_1^1, \ldots, w_1^n \leftarrow t]) \uplus$
$\quad \mathcal{V}^i(u\{e_i\langle\, w_1^i\,\rangle.w_1^i/x_i\}_{1\le i\le n}[w_1^1, \ldots, w_1^n \leftarrow t])(c)$
$\mathcal{V}^i(u\{e_i\langle\, w_1^i\,\rangle.w_1^i/x_i\}_{1\le i\le n}[w_1^1, \ldots, w_1^n \leftarrow t])(c) = \mathcal{V}^k(t)(c) = \mathcal{V}^j(t)(c)$
where $k = \mathcal{V}^i(u\{e_i\langle\, w_1^i\,\rangle.w_1^i/x_i\}_{1\le i\le n}[w_1^1, \ldots, w_1^n \leftarrow t])(w_1^1) + \ldots$
$\quad \cdots + \mathcal{V}^i(u\{e_i\langle\, w_1^i\,\rangle.w_1^i/x_i\}_{1\le i\le n}[w_1^1, \ldots, w_1^n \leftarrow t])(w_1^n) = j$
$= \mathcal{W}^i(u\{e_i\langle\, w_1^i\,\rangle.w_1^i/x_i\}_{1\le i\le n}[w_1^1, \ldots, w_1^n \leftarrow t]) \uplus \mathcal{V}^j(t)(c)$
$= \mathcal{W}^i(u\{e_i\langle\, w_1^i\,\rangle.w_1^i/x_i\}_{1\le i\le n}) \uplus \mathcal{W}^k(t) \uplus \{\mathcal{V}^j(t)(c)\}$
$= \mathcal{W}^i(u) \uplus \mathcal{W}^j(t) \uplus \{\mathcal{V}^i(u)(x_1), \ldots, \mathcal{V}^i(u)(x_n), \mathcal{V}^j(t)(c)\}$
Case: 2
$\mathcal{W}^i(u[x_1, \ldots, x_n \leftarrow c\langle\, \vec{y}\,\rangle.t]) = \mathcal{W}^i(u) \uplus \mathcal{W}^j(c\langle\, \vec{y}\,\rangle.t) = \mathcal{W}^i(u) \uplus \mathcal{W}^j(t) \uplus \{j\}$
where $j = \mathcal{V}^i(u)(x_1) + \cdots + \mathcal{V}^i(u)(x_n)$
$\mathcal{W}^i(u\{e_i\langle\, w_1^i\,\rangle.w_1^i/x_i\}_{1\le i\le n}[e_1\langle\, w_1^1\,\rangle \ldots e_n\langle\, w_1^n\,\rangle \,|\, c\langle\, \vec{y}\,\rangle\,[w_1^1, \ldots, w_1^n \leftarrow t]])$
$= \mathcal{W}^i(u\{e_i\langle\, w_1^i\,\rangle.w_1^i/x_i\}_{1\le i\le n}[w_1^1, \ldots, w_1^n \leftarrow t])$
$= \mathcal{W}^i(u\{e_i\langle\, w_1^i\,\rangle.w_1^i/x_i\}_{1\le i\le n}) \uplus \mathcal{W}^k(t)$
where $k = \mathcal{V}^i(u\{e_i\langle\, w_1^i\,\rangle.w_1^i/x_i\}_{1\le i\le n})(w_1^1) + \cdots + \mathcal{V}^i(u\{e_i\langle\, w_1^i\,\rangle.w_1^i/x_i\}_{1\le i\le n})(w_1^n) = j$
$= \mathcal{W}^i(u\{e_i\langle\, w_1^i\,\rangle.w_1^i/x_i\}_{1\le i\le n}) \uplus \mathcal{W}^j(t)$
$= \mathcal{W}^i(u) \uplus \mathcal{W}^j(t) \uplus \{\mathcal{V}^i(u)(x_1), \ldots, \mathcal{V}^i(u)(x_n)\}$

$$u[e_1\langle\, \vec{w}_1\,\rangle \ldots e_n\langle\, \vec{w}_n\,\rangle \,|\, c\langle\, c\,\rangle\,[\vec{w}_1, \ldots, \vec{w}_n \leftarrow c]] \rightsquigarrow_D u\{e_1\langle\, \vec{w}_1\,\rangle\}_e \ldots \{e_n\langle\, \vec{w}_n\,\rangle\}_e$$

$\mathcal{W}^i(u[e_1\langle\, \vec{w}_1\,\rangle \ldots e_n\langle\, \vec{w}_n\,\rangle \,|\, c\langle\, c\,\rangle\,[\vec{w}_1, \ldots, \vec{w}_n \leftarrow c]])$
$= \mathcal{W}^i(u[\vec{w}_1, \ldots, \vec{w}_n \leftarrow c]) \uplus \{\mathcal{V}^i(u[\vec{w}_1, \ldots, \vec{w}_n \leftarrow c](c))\}$
$= \mathcal{W}^i(u) \uplus \{\} \uplus \{j\}$
where $j = \mathcal{V}^i(u)(\vec{w}_1) + \cdots + \mathcal{V}^i(u)(\vec{w}_n)$
$\mathcal{W}^i(u\{e_1\langle\, \vec{w}_1\,\rangle\}_e \ldots \{e_n\langle\, \vec{w}_n\,\rangle\}_e) = \mathcal{W}^i(u) \uplus \{\mathcal{V}^i(u)(\vec{w}_1), \ldots, \mathcal{V}^i(u)(\vec{w}_n)\}$
where $\mathcal{V}^i(u)(\vec{w}) = \mathcal{V}^i(u)(w_1) + \cdots + \mathcal{V}^i(u)(w_n)$ and $\vec{w} = \{w_1, \ldots, w_n\}$ □

Lemma 60. *If $t \rightsquigarrow_{(L)} u$ then $\mathcal{W}^i(t) = \mathcal{W}^i(u)$*

Proof. We prove this case-by-case

$$u[\vec{w} \leftarrow y][\vec{x} \cdot y \leftarrow t] \rightsquigarrow_L u[\vec{x} \cdot \vec{w} \leftarrow t]$$

$\mathcal{W}^i(u[\vec{w} \leftarrow y][\vec{x} \cdot y \leftarrow t]) = \mathcal{W}^i(u[\vec{w} \leftarrow y]) \uplus \mathcal{W}^j(t)$
where $j = \mathcal{V}^i(u[\vec{w} \leftarrow y])(\vec{x}) + \mathcal{V}^i(u[\vec{w} \leftarrow y])(y) = \mathcal{V}^i(u[\vec{w} \leftarrow y])(\vec{x}) + \mathcal{V}^i(u)(\vec{w})$
$= \mathcal{W}^i(u) \uplus \mathcal{W}^j(t) = \mathcal{W}^i(u[\vec{x} \cdot \vec{w} \leftarrow t])$

$$u[x \leftarrow t] \rightsquigarrow_L u\{t/x\}$$

$\mathcal{W}^i(u[x \leftarrow t]) = \mathcal{W}^i(u) \uplus \mathcal{W}^j(t)$
where $j = \mathcal{V}^i(u)(x)$
$\mathcal{W}^i(u\{t/x\}) = \mathcal{W}^i(u) \uplus \mathcal{W}^{\mathcal{V}^i(u)(x)}(t)$

For the other lifting rules, we show that $\mathcal{V}^i(u[\Gamma])$ outputs the same integers before and after lifting for each variable bounded by $[\Gamma]$. Then we can know it produces some multiset M.

$$(s[\Gamma])\, t \rightsquigarrow_L (s\, t)[\Gamma]$$

$\mathcal{W}^i((s[\Gamma])\, t) = \mathcal{W}^i(s[\Gamma]) \uplus \mathcal{W}^i(t) = \mathcal{W}^i(s) \uplus \mathcal{W}^i(t) \uplus M_1$
$\mathcal{W}^i((s\, t)[\Gamma]) = \mathcal{W}^i(s\, t) \uplus M_2 = \mathcal{W}^i(s) \uplus \mathcal{W}^i(t) \uplus M_2$
$M_1 = M_2$ since $\mathcal{V}^i(s)(x) = \mathcal{V}^i(s\, t)(x)$ for $x \in (s)_{fv}$ and $[\Gamma]$ only binds variables in s.

$$s\, t[\Gamma] \rightsquigarrow_L (s\, t)[\Gamma]$$

This case is very similar to the one above and we omit it.

$$d\langle\, \vec{x}\, \rangle.t[\Gamma] \rightsquigarrow_L (d\langle\, \vec{x}\, \rangle.t)[\Gamma] \text{ iff all } \vec{x} \in (t)_{fv}$$

Case 1:
$\mathcal{W}^i(d\langle\, d\, \rangle.(t[\Gamma])) = \mathcal{W}^i(t[\Gamma]) \uplus \{i, \mathcal{V}^i(t[\Gamma])(d)\} = \mathcal{W}^i(t) \uplus M_1 \uplus \{i, \mathcal{V}^i(t)(d)\}$
$\mathcal{W}^i((d\langle\, d\, \rangle.t)[\Gamma]) = \mathcal{W}^i(d\langle\, d\, \rangle.t) \uplus M_2 = \mathcal{W}^i(t) \uplus M_2 \uplus \{i, \mathcal{V}^i(t)(d)\}$
$M_1 = M_2$ since $\mathcal{V}^i(t)(x) = \mathcal{W}^i(d\langle\, d\, \rangle.t)(x)$ where $x \neq d$ and d is not bound by $[\Gamma]$
Case 2:
$\mathcal{W}^i(d\langle\, \vec{x}\, \rangle.(t[\sigma])) = \mathcal{W}^i(t[\sigma]) \uplus \{i\} = \mathcal{W}^i(t) \uplus M_1 \uplus \{i\}$
$\mathcal{W}^i((d\langle\, \vec{x}\, \rangle.t)[\sigma]) = \mathcal{W}^i(d\langle\, \vec{x}\, \rangle.t) \uplus M_2 = \mathcal{W}^i(t) \uplus M_2 \uplus \{i\}$
$M_1 = M_2$ since $\mathcal{V}^i(t)(x) = \mathcal{W}^i(d\langle\, \vec{x}\, \rangle.t)(x)$ where $x \neq d$ and d is not bound by $[\Gamma]$

$$u[\vec{x} \leftarrow t[\Gamma]] \rightsquigarrow_L u[\vec{x} \leftarrow t][\Gamma]$$

Case 1:
$\mathcal{W}^i(u[\vec{x} \leftarrow t[\Gamma]]) = \mathcal{W}^i(u) \uplus \mathcal{W}^j(t[\Gamma]) = \mathcal{W}^i(u) \uplus \mathcal{W}^j(t) \uplus M_1$
where $j = \mathcal{V}^i(u)(x_1) + \cdots + \mathcal{V}^i(u)(x_n)$
$\mathcal{W}^i(u[\vec{x} \leftarrow t][\Gamma]) = \mathcal{W}^i(u[\vec{x} \leftarrow t]) \uplus M_2 = \mathcal{W}^i(u) \uplus \mathcal{W}^j(t) \uplus M_2$
$M_1 = M_2$ since $\mathcal{V}^j(t)(x) = \mathcal{V}^i(u[\vec{x} \leftarrow t])(x)$ for $x \in (t)_{fv}$ and $[\Gamma]$ only binds variables in t

Case 2:
$\mathcal{W}^i(u[\leftarrow t[\Gamma]]) = \mathcal{W}^i(u) \uplus \mathcal{W}^1(t[\Gamma]) = \mathcal{W}^i(u) \uplus \mathcal{W}^1(t) \uplus M_1$
$\mathcal{W}^i(u[\leftarrow t][\Gamma]) = \mathcal{W}^i(u[\leftarrow t]) \uplus M_2 = \mathcal{W}^i(u) \uplus \mathcal{W}^1(t) \uplus M_2$
$M_1 = M_2$ since $\mathcal{V}^1(t)(x) = \mathcal{V}^i(u[\leftarrow t])(x)$ for $x \in (t)_{fv}$ and $[\Gamma]$ only binds variables in t

$$u[e_1\langle\,\vec{w}_1\,\rangle \dots e_n\langle\,\vec{w}_n\,\rangle \,|\, c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}[\vec{y} \leftarrow t]] \rightsquigarrow_L$$
$$u\{(\vec{w}_1/\vec{y})/e_1\}_b \dots \{(\vec{w}_n/\vec{y})/e_n\}_b[e_1\langle\,\vec{w}_1/\vec{y}\,\rangle \dots e_n\langle\,\vec{w}_n/\vec{y}\,\rangle \,|\, c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}][\vec{y} \leftarrow t]$$

$$u[e_1\langle\,\vec{w}_1\,\rangle \dots e_n\langle\,\vec{w}_n\,\rangle \,|\, c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}[\overrightarrow{f\langle\,\vec{z}\,\rangle}] \,|\, d\langle\,\vec{a}\,\rangle\,\overline{[\Gamma']}]] \rightsquigarrow_L$$
$$u\{(\vec{w}_1/\vec{z})/e_1\}_b \dots \{(\vec{w}_n/\vec{z})/e_n\}_b[e_1\langle\,\vec{w}_1/\vec{z}\,\rangle \dots e_n\langle\,\vec{w}_n/\vec{z}\,\rangle \,|\, c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}][\overrightarrow{f\langle\,\vec{z}\,\rangle} \,|\, d\langle\,\vec{a}\,\rangle\,\overline{[\Gamma']}]$$

Since book-keeping operations do not affect the weight of a term (Proposition 58), we simplify these two rules into one, where u' is u with some book-keepings applied.
Note: Proposition 58 is relevant here since the book-keepings produced by this rule cannot be of the form $\{e/e\}_b$ without breaking linearity.

$$u[e_1\langle\,\vec{w}_1\,\rangle \dots e_n\langle\,\vec{w}_n\,\rangle \,|\, c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}[\Gamma]] \rightsquigarrow_L u'[e_1\langle\,\vec{z}_1\,\rangle \dots e_n\langle\,\vec{z}_1\,\rangle \,|\, c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}][\Gamma]$$

Case 1:
$\mathcal{W}^i(u[e_1\langle\,\vec{w}_1\,\rangle \dots e_n\langle\,\vec{w}_n\,\rangle \,|\, c\langle\,c\,\rangle\,\overline{[\Gamma]}[\Gamma]]) = \mathcal{W}^i(u\overline{[\Gamma]}[\Gamma]) \uplus \{\mathcal{V}^i(u\overline{[\Gamma]}[\Gamma](c))\}$
$= \mathcal{W}^i(u\overline{[\Gamma]}) \uplus M_1 \uplus \{\mathcal{V}^i(u\overline{[\Gamma]}[\Gamma](c))\}$
$\mathcal{W}^i(u'[e_1\langle\,\vec{z}_1\,\rangle \dots e_n\langle\,\vec{z}_1\,\rangle \,|\, c\langle\,c\,\rangle\,\overline{[\Gamma]}][\Gamma]) = \mathcal{W}^i(u'[e_1\langle\,\vec{z}_1\,\rangle \dots e_n\langle\,\vec{z}_1\,\rangle \,|\, c\langle\,c\,\rangle\,\overline{[\Gamma]}]) \uplus M_2$
$= \mathcal{W}^i(u'[\Gamma]) \uplus M_2 \uplus \{\mathcal{V}^i(u[\Gamma])(c)\}$
$M_1 = M_2$ since $\mathcal{V}^i(u\overline{[\Gamma]})(x) = \mathcal{V}^i(u'[e_1\langle\,\vec{z}_1\,\rangle \dots e_n\langle\,\vec{z}_1\,\rangle \,|\, c\langle\,c\,\rangle\,\overline{[\Gamma]}])(x)$
for $x \in (u\overline{[\Gamma]}/\{c, e_1, \dots, e_n\})_{fv}$ and the variables $c, e_1, \dots, e_n$ are not bound by $[\Gamma]$
$\{\mathcal{V}^i(u\overline{[\Gamma]}[\Gamma])(c)\} = \{\mathcal{V}^i(u\overline{[\Gamma]})(c)\}$ since $c \in (\overline{[\Gamma]})_{fv}$ and $\mathcal{V}^i(u\overline{[\Gamma]}[\Gamma]) = \mathcal{V}^i(u\overline{[\Gamma]}) \uplus \mathcal{V}^j([\Gamma])$.
Case 2:
$\mathcal{W}^i(u[e_1\langle\,\vec{w}_1\,\rangle \dots e_n\langle\,\vec{w}_n\,\rangle \,|\, c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}[\Gamma]]) = \mathcal{W}^i(u\overline{[\Gamma]}[\Gamma]) = \mathcal{W}^i(u\overline{[\Gamma]}) \uplus M_1$
$\mathcal{W}^i(u'[e_1\langle\,\vec{z}_1\,\rangle \dots e_n\langle\,\vec{z}_1\,\rangle \,|\, c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}][\Gamma]) = \mathcal{W}^i(u'[e_1\langle\,\vec{z}_1\,\rangle \dots e_n\langle\,\vec{z}_1\,\rangle \,|\, c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}]) \uplus M_2$
$= \mathcal{W}^i(u'\overline{[\Gamma]}) \uplus M_2$
$M_1 = M_2$ since $\mathcal{V}^i(u\overline{[\Gamma]})(x) = \mathcal{V}^i(u'[e_1\langle\,\vec{z}_1\,\rangle \dots e_n\langle\,\vec{z}_1\,\rangle \,|\, c\langle\,c\,\rangle\,\overline{[\Gamma]}])(x)$
for $x \in (u\overline{[\Gamma]}/\{c, e_1, \dots, e_n\})_{fv}$ and the variables $c, e_1, \dots, e_n$ are not bound by $[\Gamma]$ □

We now prove that for the remaining reduction rules, the number of closures decreases, otherwise it remains constant. These are rules (c_1) and (c_2).

Lemma 61. *For $u \rightsquigarrow_{(C)} t$, the weights $\mathcal{W}^i(u) = \mathcal{W}^i(t)$.*

Proof. We prove this case-by-case

$$u[\vec{w} \leftarrow y][\vec{x} \cdot y \leftarrow t] \rightsquigarrow_C u[\vec{x} \cdot \vec{w} \leftarrow t]$$

$\mathcal{W}^i(u[\vec{w} \leftarrow y][\vec{x} \cdot y \leftarrow t]) = \mathcal{W}^i(u[\vec{w} \leftarrow y]) \uplus \mathcal{W}^j(t)$
where $j = \mathcal{V}^i(u[\vec{w} \leftarrow y])(\vec{x}) + \mathcal{V}^i(u[\vec{w} \leftarrow y])(y) = \mathcal{V}^i(u[\vec{w} \leftarrow y])(\vec{x}) + \mathcal{V}^i(u)(\vec{w})$

$= \mathcal{W}^i(u) \uplus \mathcal{W}^j(t) = \mathcal{W}^i(u[\vec{x} \cdot \vec{w} \leftarrow t])$

$$u[x \leftarrow t] \rightsquigarrow_C u\{t/x\}$$

$\mathcal{W}^i(u[x \leftarrow t]) = \mathcal{W}^i(u) \uplus \mathcal{W}^j(t)$
where $j = \mathcal{V}^i(u)(x)$
$\mathcal{W}^i(u\{t/x\}) = \mathcal{W}^i(u) \uplus \mathcal{W}^{\mathcal{V}^i(u)(x)}(t)$ □

5.3 Strong normalisation and confluence of $\rightsquigarrow_{(R,D,L,C)}$

We are now ready to prove that there does not exists an infinite reduction sequence consisting of only sharing reductions ($\rightsquigarrow_{(R,D,L,C)}$). We first define the *sharing measure* of a term.

Definition 62. *The* sharing measure *of a Λ^S_a-term t is a triple ($\mathcal{W}(t)$, C, $\mathcal{H}(t)$), where C is the number of closures in t. We compare sharing measures by using the lexicographical preferences according to $\mathcal{W} > C > \mathcal{H}$.*

We show strong normalisation by showing that the sharing measure decreases during reduction. Lemma 60 says that the weight remains unchanged during $\rightsquigarrow_L$ reductions and Lemma 61 states the weight remains unchanged during $\rightsquigarrow_C$ reductions, and Lemma 59 says that the weight strictly decreases during $\rightsquigarrow_{R,D}$ reduction. Therefore there exists a minimal weight in the reduction, where the weight cannot decrease anymore. Once we reach this minimal weight, we won't be able to perform any more $\rightsquigarrow_{(D,R)}$ reduction steps. Therefore, if we can perform any sharing reductions, it will be $\rightsquigarrow_{L,C}$ which does not affect the weight. The $\rightsquigarrow_C$ reductions strictly decrease the number of closures in a term, which remains unaffected by $\rightsquigarrow_L$. Therefore a minimal number of closures exists. After this, we are left in a situation where if we can perform further reductions, they are $\rightsquigarrow_L$ reductions which strictly decrease the height measure, while leaving the weight and number of closures unchanged. Therefore, a minimal height measure will then eventually be obtained, where we can no longer perform any $\rightsquigarrow_L$ steps. This is when we reach sharing normal form.

Theorem 63. *Sharing reduction $\rightsquigarrow_{(R,D,L,C)}$ is strongly normalising*

Proof. From Lemma 59, Lemma 60, Lemma 61 and Lemma 54, it follows that the sharing measure of a term is strictly decreasing under $\rightsquigarrow_{(R,D,L,C)}$, proving the statement. □

Now that we have proven the sharing reductions are strongly normalising, we can prove that they are confluent for closed terms.

Theorem 64. *The sharing reduction relation $\rightsquigarrow_{(R,D,L,C)}$ is confluent*

Proof. Lemma 25 tells us that the preservation is preserved under reduction i.e. for $s \rightsquigarrow_{(R,D,L,C)} t$, $[\![s]\!] = [\![t]\!]$. Therefore given $t \rightsquigarrow^*_{(R,D,L,C)} s_1$ and $t \rightsquigarrow^*_{(R,D,L,C)} s_2$, $[\![t]\!] = [\![s_1]\!] = [\![s_2]\!]$. Since we know that sharing reductions are strongly normalising, we know there exists terms u_1 and u_2 in sharing normal form such that $s_1 \rightsquigarrow^*_{(R,D,L,C)} u_1$ and $s_2 \rightsquigarrow^*_{(R,D,L,C)} u_2$. Lemma

24 tells us that terms in terms in sharing normal form are in correspondence with their denotations i.e. $(\!|[\![t]\!]|\!) = t$. Since by Lemma 25 we know $[\![u_1]\!] = [\![s_1]\!] = [\![s_2]\!] = [\![u_2]\!]$, and by Lemma 24 $(\!|[\![u_1]\!]|\!) = u_1$ and $(\!|[\![u_2]\!]|\!) = u_2$, we can conclude $u_1 = u_2$. Hence, we prove confluence. □

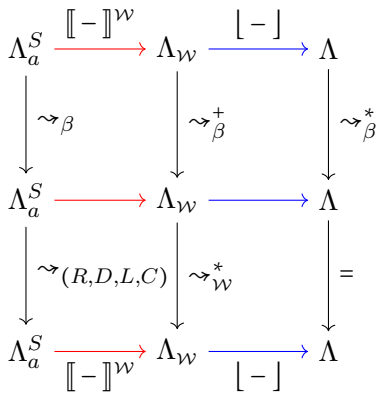
Chapter 6

Preservation of Strong Normalisation and Confluence

善始善终
shànshǐ shànzhōng
To start well and to end well

6.1 Preservation of Strong Normalisation

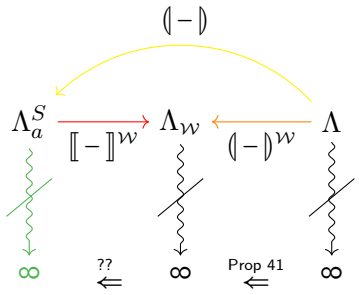
Here we show how Λ_a^S preserves strong normalisation with respect to the λ-calculus, using the results from the previous chapters. Observe the following diagram



We obtain this diagram as a result of Lemma 43 and Lemma 44, and Lemma 25.

Recall that by Proposition 39 that for all $N \in \Lambda$, $[\![(\!| N |\!)]\!]^{\mathcal{W}} = (\!| N |\!)^{\mathcal{W}}$, and that Proposition 41 states if a term $N \in \Lambda$ is strongly normalising then so is $(\!| N |\!)^{\mathcal{W}}$. Observe that the statement 'if term M has an infinite reduction sequence then term N has an infinite reduction sequence' is equivalent to 'if term N is strongly normalising then term M is strongly normalising' by contraposition.

Therefore, given a strongly normalising term $N \in \Lambda$, we know that its corresponding weakening term is also strongly normalising. Furthermore, since $[\![(\!| N |\!)]\!]^{\mathcal{W}} = (\!| N |\!)^{\mathcal{W}}$, we know that $[\![(\!| N |\!)]\!]^{\mathcal{W}}$ is also strongly normalising.



We prove that the spinal λ-calculus preserves strong normalisation with the following Lemma.

Lemma 65. *For $t \in \Lambda_a^S$ has an infinite reduction path, then $[\![t]\!]^{\mathcal{W}}$ also has an infinite reduction path.*

Proof. Due to Theorem 64, we know that the infinite reduction path contains an infinite β-reduction. This means in the reduction sequence, between each β-reduction, there are finite many $\rightsquigarrow_{(R,D,L,C)}$ reduction steps. Lemma 44 says each $\rightsquigarrow_{(R,D,L,C)}$ step in Λ_a^S corresponds to zero or more weakening reductions ($\rightsquigarrow_{\mathcal{W}}^*$). Lemma 43 says that each beta reduction in Λ_a^S corresponds to one or more β-steps in $\Lambda_{\mathcal{W}}$. Therefore, it is inevitable that $[\![t]\!]^{\mathcal{W}}$ also has an infinite reduction path. □

Theorem 66. *If $N \in \Lambda$ is strongly normalising, then so is $(|N|)$.*

Proof. For a given $N \in \Lambda$ that is strongly normalising, we know by Lemma 41 that $(|N|)^{\mathcal{W}}$ is strongly normalising. Then $[\![(|N|)]\!]^{\mathcal{W}}$ is strongly normalising, since Proposition 39 states that $(|N|)^{\mathcal{W}} = [\![(|N|)]\!]^{\mathcal{W}}$. Then by Lemma 65, which states that if $[\![t]\!]^{\mathcal{W}}$ is strongly normalising, then t is strongly normalising, proves that $(|N|)$ is strongly normalising. □

6.2 Confluence

We use the proofs used to prove preservation of strong normalisation to also prove confluence of Λ_a^S-terms. We will also use the folllowing Lemma.

Lemma 67. *[Simulation] Given $N, M \in \Lambda$. If $N \rightsquigarrow_\beta M$, then we have*

$$(|N|) \rightsquigarrow_\beta \rightsquigarrow_{(R,D,L,C)}^* (|M|)$$

Proof. We prove this by induction on N. Recall that $\frac{n}{x}$ means replacing x with n fresh distinct variables. Let σ be the substitutions for $\frac{n_1}{x_1} \dots \frac{n_m}{x_m}$ where $x_1, \dots, x_m$ occur free in N, replacing the different occurrences of each free variable with fresh, distinct variables. Let $\overline{[\Gamma]} = [\vec{x_1} \leftarrow x_1] \dots [\vec{x_m} \leftarrow x_m]$.

Base Case: $(\lambda x.N)\, M \rightsquigarrow_\beta N\{M/x\}$

SubCase: $|N|_x = 1$
$(|(\lambda x.N)\, M|) = ((x\langle x \rangle.(|N\sigma|)')\, (|M\sigma|)')\overline{[\Gamma]} \rightsquigarrow_\beta (|N\sigma|)'\{(|M\sigma|)'/x\}\overline{[\Gamma]} = (|N\{M/x\}|)$

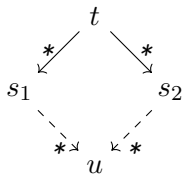
SubCase: $|U|_x = m$
$(\!|(\lambda x.N)\,M|\!) = ((x\langle x\rangle.(\!|N\sigma|\!)'[\vec{x} \leftarrow x])\,(\!|M\sigma|\!)')\overline{[\Gamma]}$
$\rightsquigarrow_\beta (\!|N\sigma|\!)'[\vec{x} \leftarrow (\!|M\sigma|\!)'][\overline{\Gamma}] = (\!|N\{M/x\}|\!)$

Inductive Case: Application $N\,M \rightsquigarrow_\beta N'\,M$
$(\!|N\,M|\!) = ((\!|N\sigma|\!)'\,(\!|M\sigma|\!)')\overline{[\Gamma]}$
by induction hypothesis
$\rightsquigarrow_\beta ((\!|N'\sigma|\!)'\,(\!|M\sigma|\!)')\overline{[\Gamma]} = (\!|N'\,M|\!)$

Inductive Case: Abstraction $\lambda x.N \rightsquigarrow \lambda x.N'$
$(\!|\lambda x.N|\!) = (x\langle x\rangle.(\!|N\sigma|\!)')\overline{[\Gamma]}$
by induction hypothesis
$\rightsquigarrow_\beta = (x\langle x\rangle.(\!|N'\sigma|\!)')\overline{[\Gamma]} = (\!|\lambda x.N'|\!)$ □

Confluence was proven for the λ-calculus in 1936 by Church and Rosser in [CR36]. Using these results, we can show confluence for Λ_a^S-terms.

Theorem 68. *Given $t, s_1, s_2 \in \Lambda_a^S$. If $t \rightsquigarrow^*_{(\beta,R,D,L,C)} s_1$ and $t \rightsquigarrow^*_{(\beta,R,D,L,C)} s_2$, there exists a $u \in \Lambda_a^S$ such that $s_1 \rightsquigarrow^*_{(\beta,R,D,L,C)} u$ and $s_2 \rightsquigarrow^*_{(\beta,R,D,L,C)} u$.*



Proof. Suppose $t \rightsquigarrow^*_{(\beta,R,D,L,C)} s_1$ and $t \rightsquigarrow^*_{(\beta,R,D,L,C)} s_2$. Then by Lemma 25 we have $[\![t]\!] \rightsquigarrow^*_\beta [\![s_1]\!]$ and $[\![t]\!] \rightsquigarrow^*_\beta [\![s_2]\!]$. By the Church-Rosser theorem [CR36], there exists a $M \in \Lambda$ such that $[\![s_1]\!] \rightsquigarrow^*_\beta M$ and $[\![s_2]\!] \rightsquigarrow^*_\beta M$. Due to Lemma 26, $(\!|[\![s_1]\!]|\!) = s_1'$ and $(\!|[\![s_2]\!]|\!) = s_2'$ where $s_1', s_2' \in \Lambda_a^S$ in sharing normal form. Then thanks to Lemma 67 we know $s_1' \rightsquigarrow^*_{(\beta,R,D,L,C)} (\!|M|\!)$ and $s_2' \rightsquigarrow^*_{(\beta,R,D,L,C)} (\!|M|\!)$. Combined, we get confluence. □

Chapter 7

Spine Duplication

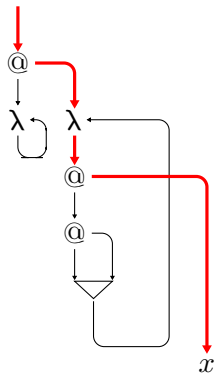
由简及繁
yóujiǎn jífán
From the simple to the complex

In this chapter we prove that our calculus is capable of performing spine duplication by showing the spine of any abstraction-term and only the spine can be duplicated. This spine here is not the same spine used in [BKKS87], which defines the spine to be the path towards the head variable.

We formalise the definition of the spine of a term in the spinal atomic λ-calculus, and show that it is a reasonable definition when considering the equivalent spine in the λ-calculus. We then show that the reduction rules $\rightsquigarrow_{(R,D,L,C)}$ are capable of duplicating only the spine of the term, and any subterm that is not part of the spine is remained shared.

7.1 The $\mathcal{V}$-Spine

We first discuss spines in the λ-calculus. Intuitively, given a set of variables $\mathcal{V}$ and $M \in \Lambda$, the spines is composed of the shortest paths from the root of the term to the free variables of M in $\mathcal{V}$. The spine can also be seen as the union of the paths to variable occurrences described by director strings [FMS05b], and what is duplicated in the implementation of [BLM07]. To help with intuition, let $M = (\lambda z.z)\,\lambda y.(y\,y)\,x$, then the x-spine of M ($\text{spine}_x(M)$) is thick red line in the graph below.



Definition 69. *Given a term $M \in \Lambda$, and a set of variables V that do not occur bound in M, the V-spine of M is defined inductively as follows (where y is a fresh variable)*

$$\mathit{spine}_V(x) = x$$

$$\mathit{spine}_V(\lambda x.M) = \begin{cases} y & (M)_{fv} \cap V = \{\} \\ \lambda x.\mathit{spine}_V(M) & \mathit{otherwise} \end{cases}$$

$$\mathit{spine}_V(M\,N) = \begin{cases} y & (M\,N)_{fv} \cap V = \{\} \\ (\mathit{spine}_V(M))\,y & (N)_{fv} \cap V = \{\} \\ (y)\,\mathit{spine}_V(N) & (M)_{fv} \cap V = \{\} \\ (\mathit{spine}_V(M))\,\mathit{spine}_V(N) & \mathit{otherwise} \end{cases}$$

Substitution commutes with the function determining the spine in the following way

Proposition 70. *Given $M, N \in \Lambda$, $\mathit{spine}_{V\cup\{x\}}(M)\{\mathit{spine}_V(N)/x\} = \mathit{spine}_V(M\{N/x\})$*

Proof. We prove this by induction on M

Base Case: Variable
$\text{spine}_{V\cup\{x\}}(x)\{\text{spine}_V(N)/x\} = x\{\text{spine}_V(N)/x\} = \text{spine}_V(N) = \text{spine}_V(x\{N/x\})$

Base Case: Variable
$\text{spine}_{V\cup\{x\}}(y)\{\text{spine}_V(N)/x\} = y\{\text{spine}_V(N)/x\} = y = \text{spine}_V(y) = \text{spine}_V(y\{N/x\})$

Inductive Case: Application

$\mathsf{spine}_{V\cup\{x\}}(M_1\,M_2)\{\mathsf{spine}_V(N)/x\}$
Case: $\mathsf{spine}_{V\cup\{x\}}(M_1\,M_2) = y$
$= y\{\mathsf{spine}_V(N)/x\} = y = \mathsf{spine}_V(M_1\,M_2)$

Case: $\mathsf{spine}_{V\cup\{x\}}(M_1\,M_2) = \mathsf{spine}_{V\cup\{x\}}(M_1)\,y$
$= \mathsf{spine}_{V\cup\{x\}}(M_1)\,y\{\mathsf{spine}_V(N)/x\} = \mathsf{spine}_{V\cup\{x\}}(M_1)\{\mathsf{spine}_V(N)/x\}\,y$
$\overset{\text{I.H.}}{=} \mathsf{spine}_V(M_1\{N/x\})\,y = \mathsf{spine}_V(M_1\{N/x\}\,M_2\{N/x\}) = \mathsf{spine}_V((M_1\,M_2)\{N/x\})$

Case: $\mathsf{spine}_{V\cup\{x\}}(M_1\,M_2) = y\,\mathsf{spine}_{V\cup\{x\}}(M_2)$
Similar to the previous case and we omit the proof

Case: $\mathsf{spine}_{V\cup\{x\}}(M_1\,M_2) = (\mathsf{spine}_{V\cup\{x\}}(M_1))\,\mathsf{spine}_{V\cup\{x\}}(M_2)$
$= (\mathsf{spine}_{V\cup\{x\}}(M_1))\,\mathsf{spine}_{V\cup\{x\}}(M_2)\{\mathsf{spine}_V(N)/x\}$
$= (\mathsf{spine}_{V\cup\{x\}}(M_1)\{\mathsf{spine}_V(N)/x\})\,\mathsf{spine}_{V\cup\{x\}}(M_2)\{\mathsf{spine}_V(N)/x\}$
$\overset{\text{I.H.}}{=} (\mathsf{spine}_V(M_1\{N/x\}))\,\mathsf{spine}_V(M_2\{N/x\}) = \mathsf{spine}_V((M_1\,M_2)\{N/x\})$

Inductive Case: Abstraction
$\mathsf{spine}_{V\cup\{x\}}(\lambda z.M)\{\mathsf{spine}_V(N)/x\}$
Case: $\mathsf{spine}_{V\cup\{x\}}(\lambda z.M) = y$
$= y\{\mathsf{spine}_V(N)/x\} = y = \mathsf{spine}_V((\lambda z.M)\{N/x\})$

Case: $\mathsf{spine}_{V\cup\{x\}}(\lambda z.M) = \lambda z.\mathsf{spine}_V(M)$
$= (\lambda z.\mathsf{spine}_V(M))\{\mathsf{spine}_V(N)/x\} = \lambda z.\mathsf{spine}_V(M)\{\mathsf{spine}_V(N)/x\}$
$\overset{\text{I.H.}}{=} \lambda z.\mathsf{spine}_V(M\{N/x\}) = \mathsf{spine}_V((\lambda z.M)\{N/x\})$ □

We now define the spine in terms of the spinal atomic λ-calculus. In order to deal with the sharing and distribtuor constructs, we need to duplicate shared terms (collapsing sharings). Therefore, we introduce *variants* of terms.

Definition 71. *A* variant *of a term t is a term obtained from t by renaming all (bound or free) variables. A variant is* fresh *if its variables are fresh variables.*

Notation 72. *We use t^i to denote a fresh variant of t obtained by replacing each variable x in t by a fresh one x^i.*

Additionally, when determining the V-spine of a term in Λ^S_a, we may need to update the variables in phantom-abstractions as the spine may introduce fresh variables. To keep track of the variables, we define an auxiliary function *update book-keeping* which compares two terms (one being the spine and one being the original term), and determine what subterms containing variables in the tuple of a phantom-abstraction have been replaced with which fresh variables.

Definition 73. *We define the* update book-keeping *function* $UBK(V\,,\,\Lambda^S_a\,,\,\Lambda^S_a) \to V^*$ *as*

$$
\begin{aligned}
UBK(x\,,\,t\,,\,y) &= \{y\}\\
UBK(x\,,\,c\langle\,\vec{y}\,\rangle.t\,,\,c\langle\,\vec{z}\,\rangle.t') &= UBK(x\,,\,t\,,\,t')\\
UBK(x\,,\,s\,t\,,\,s'\,t') &= \begin{cases} UBK(x\,,\,s\,,\,s') & x \in (s)_{fv}\\ UBK(x\,,\,t\,,\,t') & \textit{otherwise}\end{cases}\\
UBK(x\,,\,s[z_1,\ldots,z_n \leftarrow t]\,,\,u) &= \begin{cases} \bigcup\limits_{j\le n} UBK(x_j\,,\,s\{t^i/z_i\}_{i\le n}\,,\,u) & x \in (t)_{fv}\\ UBK(x\,,\,s\,,\,u) & \textit{otherwise}\end{cases}\\
UBK(x\,,\,t[\overrightarrow{e\langle\,\vec{w}\,\rangle}\,|\,c\langle\,\vec{y}\,\rangle\,\overline{[\Gamma]}]\,,\,u) &= UBK(x\,,\,t\overline{[\Gamma]}\,,\,u)
\end{aligned}
$$

Definition 74 (V-Spine)**.** *The V-spine $\mathit{spine}_V(t)$ of a FALC term t, with V a set of variables that do not occur bound in t, is defined inductively as follows (where y is a fresh variable).*

$$
\begin{aligned}
\mathit{spine}_V(x) &= x\\
\mathit{spine}_V(c\langle\,\vec{x}\,\rangle.t) &= y\\
&\quad \textit{if } (t)_{fv} \cap V = \{\}\\
\mathit{spine}_V(c\langle\,\vec{x}\,\rangle.t) &= c\langle\,\vec{y}\,\rangle.\mathit{spine}_V(t)\\
&\quad \textit{where } \{\vec{y}\} = \bigcup_{x\in\vec{x}} UBK(x\,,\,t\,,\,\mathit{spine}_V(t))\\
\mathit{spine}_V(u[\overrightarrow{e\langle\,\vec{w}\,\rangle}\,|\,c\langle\,\vec{x}\,\rangle\,\overline{[\Gamma]}]) &= \mathit{spine}_V(u\overline{[\Gamma]})
\end{aligned}
$$

$$
\mathit{spine}_V(s\,t) = \begin{cases} y & \textit{if } (s\,t)_{fv} \cap V = \{\};\ \textit{otherwise}\\ (\mathit{spine}_V(s))\,y & \textit{if } (t)_{fv} \cap V = \{\}\\ (y)\,\mathit{spine}_V(t) & \textit{if } (s)_{fv} \cap V = \{\}\\ (\mathit{spine}_V(s))\,\mathit{spine}_V(t) & \textit{otherwise}\end{cases}
$$

$$
\mathit{spine}_V(s[x_1\ldots x_n \leftarrow t]) = \begin{cases} y & \textit{if } (s[x_1\ldots x_n \leftarrow t])_{fv} \cap V = \{\}\\ (\mathit{spine}_{V\cup U}(s\,\sigma)) & \textit{otherwise}\end{cases}
$$

where σ are the substitutions $\{t^i/x_i\}_{1\le i\le n}$
and U is such that $z \in (V \cap (t)_{fv}) \Longleftrightarrow \forall_{1\le i\le n} z_i \in U$

Proposition 75. *Given $t \in \Lambda^S_a$, $\mathit{spine}_V(t)^i = \mathit{spine}_{V^i}(t^i)$ where $v \in V \Longleftrightarrow v_i \in V^i$*

Example 76. *Let $t = x\langle\,x\,\rangle.y\langle\,y\,\rangle.(z_1\,x)\,(z_2\,(y_1\,(y_2\,w)[y_1, y_2 \leftarrow y]))[z_1, z_2 \leftarrow z]$.*

$\mathit{spine}_{\{z\}}(t) = x\langle\,a\,\rangle.y\langle\,b\,\rangle.(z_1\,a)\,(z_2\,b)$

$\mathit{spine}_{\{w\}}(t) = x\langle\, a \,\rangle.y\langle\, c, d \,\rangle.a\,(b\,(c\,(d\,w)))$

$\mathit{spine}_{\{w,z\}}(t) = x\langle\, a \,\rangle.y\langle\, b, c \,\rangle.(z_1\,a)\,(z_2\,(b\,(c\,w)))$

7.2 Spine Equivalence

To show we have a good definition of spine for the spinal atomic λ-calculus, we want to show that the V-spine of $t \in \Lambda_a^S$ is the same as the V-spine of $M = [\![\, t \,]\!]$. When calculating the spine of $t \in \Lambda_a^S$, we may duplicate the spine of shared terms. To avoid breaking linearity, we create fresh variants for each duplicate. To show that this spine corresponds to the spine of $[\![\, t \,]\!] = M \in \Lambda$, we need to replace each variant with its original variable name.

In this section, we temporarily modify the definition of a spine for Λ_a^S-terms a little. Specifically the case for sharing. The spine of a Λ_a^S-term needs to be a Λ_a^S-term itself, thus Definition 74 collapses sharings and creates fresh variants of the term being shared. This is to maintain linearity. This constraint is not required for interpretation of Λ_a^S-terms into Λ-terms $[\![- \mid \frac{=}{-}]\!]$ (Definition 13). Therefore we use to following definition below to prove equivalence, which is an alteration of Definition 74.

Definition 77.

$$\mathit{spine}_V(s[x_1 \dots x_n \leftarrow t]) = \begin{cases} y & \text{if } (s[x_1 \dots x_n \leftarrow t])_{fv} \cap V = \{\} \\ (\mathit{spine}_V(s\,\sigma)) & \text{otherwise} \end{cases}$$

where σ are the substitutions $\{t/x_i\}_{1 \le i \le n}$

Therefore, in this section we prove the equivalence of the spine of $t \in \Lambda_a^S$ and the spine of $[\![\, t \,]\!] \in \Lambda$, by using the altered definition below for the case of sharing which does not introduce fresh variants. If these spines are equal, then the spines when using Definition 74 only differ by variable names. Intuitively, these are the variables that would be captured by some sharing that shares the original variable name during reduction.

Proposition 78. *Using the definition of V-spine from Definition 69 and Definition 77, and the readback interpretation from Definition 13, we have the following equivalence.*

$$\mathit{spine}_V([\![\, t \mid \frac{I}{\gamma} \,]\!]) = [\![\, \mathit{spine}_V(t) \mid \frac{I}{\gamma} \,]\!]$$

Proof. We prove this by induction on the sharing weight of t i.e. $\mathcal{W}^1(t)$

Base Case: Variable
$\mathsf{spine}_V([\![\, x \mid \frac{I}{\gamma} \,]\!]) = \mathsf{spine}_V(x) = x = [\![\, x \mid \frac{I}{\gamma} \,]\!] = [\![\, \mathsf{spine}_V(x) \mid \frac{I}{\gamma} \,]\!]$

Inductive Case: Application

SubCase: $\mathsf{spine}_V(s\,t) = y$
$\mathsf{spine}_V([\![\, s\,t \mid \frac{I}{\gamma} \,]\!]) = \mathsf{spine}_V([\![\, s \mid \frac{I}{\gamma} \,]\!]\,[\![\, t \mid \frac{I}{\gamma} \,]\!]) = y = [\![\, y \mid \frac{I}{\gamma} \,]\!] = [\![\, \mathsf{spine}_V(s\,t) \mid \frac{I}{\gamma} \,]\!]$

SubCase: $\mathsf{spine}_V(s\,t) = \mathsf{spine}_V(s)\,y$
$\mathsf{spine}_V([\![\, s\,t \mid \frac{I}{\gamma}\,]\!]) = \mathsf{spine}_V([\![\, s \mid \frac{I}{\gamma}\,]\!]\,[\![\, t \mid \frac{I}{\gamma}\,]\!]) = \mathsf{spine}_V([\![\, s \mid \frac{I}{\gamma}\,]\!])\,y$
$\overset{\text{I.H.}}{=} [\![\, \mathsf{spine}_V(s) \mid \frac{I}{\gamma}\,]\!]\,y = [\![\, \mathsf{spine}_V(s)\,y \mid \frac{I}{\gamma}\,]\!] = [\![\, \mathsf{spine}_V(s\,t) \mid \frac{I}{\gamma}\,]\!]$

SubCase: $\mathsf{spine}_V(s\,t) = y\,\mathsf{spine}_V(s)$
This case is similar as before and we omit it

SubCase: $\mathsf{spine}_V(s\,t) = \mathsf{spine}_V(t)\,\mathsf{spine}_V(s)$
$\mathsf{spine}_V([\![\, s\,t \mid \frac{I}{\gamma}\,]\!]) = \mathsf{spine}_V([\![\, s \mid \frac{I}{\gamma}\,]\!])\,\mathsf{spine}_V([\![\, t \mid \frac{I}{\gamma}\,]\!])$
$\overset{\text{I.H.}}{=} [\![\, \mathsf{spine}_V(s) \mid \frac{I}{\gamma}\,]\!]\,[\![\, \mathsf{spine}_V(t) \mid \frac{I}{\gamma}\,]\!] = [\![\, \mathsf{spine}_V(s\,t) \mid \frac{I}{\gamma}\,]\!]$

Inductive Case: Abstraction
$\mathsf{spine}_V([\![\, x\langle\, x\,\rangle.t \mid \frac{I}{\gamma}\,]\!]) = \lambda x.\mathsf{spine}_V([\![\, t \mid \frac{I}{\gamma}\,]\!]) \overset{\text{I.H.}}{=} \lambda x.[\![\, \mathsf{spine}_V(t) \mid \frac{I}{\gamma}\,]\!] = [\![\, \mathsf{spine}_V(x\langle\, x\,\rangle.t) \mid \frac{I}{\gamma}\,]\!]$

Inductive Case: Phantom-Abstraction
$\mathsf{spine}_V([\![\, c\langle\, x_1, \ldots, x_n\,\rangle.t \mid \frac{I}{\gamma}\,]\!]) = \lambda c.\mathsf{spine}_V([\![\, t \mid \frac{I}{\gamma}\,]\!])$
$= \lambda c.[\![\, \mathsf{spine}_V(t) \mid \frac{I}{\gamma}\,]\!] = [\![\, \mathsf{spine}_V(c\langle\, x_1, \ldots, x_n\,\rangle.t) \mid \frac{I}{\gamma}\,]\!]$

Inductive Case: Sharing
$\mathsf{spine}_V([\![\, t[x_1, \ldots, x_n \leftarrow s] \mid \frac{I}{\gamma}\,]\!]) = \mathsf{spine}_V([\![\, t \mid \frac{\sigma}{\gamma}\,]\!]) \overset{\text{prop ??}}{=} \mathsf{spine}_V([\![\, t\sigma \mid \frac{I}{\gamma}\,]\!])$
$\overset{\text{I.H.}}{=} [\![\, \mathsf{spine}_V(t\sigma) \mid \frac{I}{\gamma}\,]\!] \overset{\text{Def 77}}{=} [\![\, \mathsf{spine}_V(t[x_1, \ldots, x_n \leftarrow s]) \mid \frac{I}{\gamma}\,]\!]$

Inductive Case: Distributor
$\mathsf{spine}_V([\![\, u[\overrightarrow{e\langle\, \vec{w}\,\rangle} \mid c\langle\, \vec{x}\,\rangle\,\overline{[\Gamma]}] \mid \frac{I}{\gamma}\,]\!])$
SubCase: $\vec{x} = c$
$\mathsf{spine}_V([\![\, u[\overrightarrow{e\langle\, \vec{w}\,\rangle} \mid c\langle\, c\,\rangle\,\overline{[\Gamma]}] \mid \frac{I}{\gamma}\,]\!]) = \mathsf{spine}_V([\![\, u\overline{[\Gamma]} \mid \frac{I}{\gamma'}\,]\!])$
$\overset{\text{I.H.}}{=} [\![\, u\overline{\Gamma} \mid \frac{I}{\gamma'}\,]\!] = [\![\, \mathsf{spine}_V(u[\overrightarrow{e\langle\, \vec{w}\,\rangle} \mid c\langle\, c\,\rangle\,\overline{[\Gamma]}]) \mid \frac{I}{\gamma}\,]\!]$

SubCase: $\vec{x} = x_1, \ldots, x_m$
$\mathsf{spine}_V([\![\, u[\overrightarrow{e\langle\, \vec{w}\,\rangle} \mid c\langle\, x_1, \ldots, x_m\,\rangle\,\overline{[\Gamma]}] \mid \frac{I}{\gamma}\,]\!]) = \mathsf{spine}_V([\![\, u\overline{[\Gamma]} \mid \frac{I}{\gamma'}\,]\!])$
$\overset{\text{I.H.}}{=} [\![\, u\overline{\Gamma} \mid \frac{I}{\gamma'}\,]\!] = [\![\, \mathsf{spine}_V(u[\overrightarrow{e\langle\, \vec{w}\,\rangle} \mid c\langle\, x_1, \ldots, x_m\,\rangle\,\overline{[\Gamma]}]) \mid \frac{I}{\gamma}\,]\!]$ □

7.3 Full Laziness

In this section we show that the calculus is capable of performing spine duplication, i.e. when duplicating an abstraction, we can duplicate the spine of the term and leave all maximal subterms that do not have the spine as a subterm shared.

SubLemma 79. *Given $t, s \in \Lambda_a^S$, $\mathsf{spine}_V(t\{\mathsf{spine}_V(s)/x\}) = \mathsf{spine}_V(t\{s/x\})$*

Proof. We prove this by induction on t

Base Case: Variable
$\mathsf{spine}_V(x\{\mathsf{spine}_V(s)/x\}) = \mathsf{spine}_V(\mathsf{spine}_V(s)) = \mathsf{spine}_V(s) = \mathsf{spine}_V(x\{s/x\})$

$\mathsf{spine}_V(y\{\mathsf{spine}_V(s)/x\}) = \mathsf{spine}_V(y) = \mathsf{spine}_V(y\{s/x\})$

Inductive Case: Application
$\mathsf{spine}_V(u\,t\{\mathsf{spine}_V(s)/x\}) = (\mathsf{spine}_V(u\{\mathsf{spine}_V(s)/x\}))\,\mathsf{spine}_V(t\{\mathsf{spine}_V(s)/x\})$
$\overset{\text{I.H.}}{=} (\mathsf{spine}_V(u\{s/x\}))\,\mathsf{spine}_V(t\{s/x\}) = \mathsf{spine}_V((u\,t)\{s/x\})$

Inductive Case: Abstraction
$\mathsf{spine}_V((c\langle\,\vec{c}\,\rangle.t)\{\mathsf{spine}_V(s)/x\}) = \mathsf{spine}_V(c\langle\,\vec{c}\,\rangle.t\{\mathsf{spine}_V(s)/x\})$
$= c\langle\,\vec{e}\,\rangle.\mathsf{spine}_V(t\{\mathsf{spine}_V(s)/x\}) \overset{\text{I.H.}}{=} c\langle\,\vec{e}\,\rangle.\mathsf{spine}_V(t\{s/x\}) = \mathsf{spine}_V(c\langle\,\vec{c}\,\rangle.t\{s/x\})$

Inductive Case: Sharing
$\mathsf{spine}_V(u[z_1,\ldots,z_n \leftarrow t]\{\mathsf{spine}_V(s)/x\})$
$= \mathsf{spine}_V(u\{\mathsf{spine}_V(s)/x\}[z_1,\ldots,z_n \leftarrow t\{\mathsf{spine}_V(s)/x\}])$
$= \mathsf{spine}_V(u\{\mathsf{spine}_V(s)/x\}\{(t\{\mathsf{spine}_V(s)/x\})^i/z_i\}_{1\le i\le n})$
$\overset{\text{I.H.}}{=} \mathsf{spine}_V(u\{s/x\}\{(t\{s/x\})^i/z_i\}_{1\le i\le n})$
$= \mathsf{spine}_V(u\{s/x\}[z_1,\ldots,z_n \leftarrow t\{s/x\}])$
$= \mathsf{spine}_V(u[z_1,\ldots,z_n \leftarrow t]\{s/x\})$

Inductive Case: Distributor
$\mathsf{spine}_V(u[e_1\langle\,\vec{w_1}\,\rangle\ldots e_m\langle\,\vec{w_m}\,\rangle \mid c\langle\,\vec{y}\,\rangle\,\overline{[\Gamma]}]\{\mathsf{spine}_V(s)/x\})$
$= \mathsf{spine}_V(u[e_1\langle\,\vec{w_1}\,\rangle\ldots e_m\langle\,\vec{w_m}\,\rangle \mid c\langle\,\vec{y}\,\rangle\,\overline{[\Gamma]}\{\mathsf{spine}_V(s)/x\}])$
$= \mathsf{spine}_V(u\overline{[\Gamma]}\{\mathsf{spine}_V(s)/x\}) \overset{\text{I.H.}}{=} \mathsf{spine}_V(u\overline{[\Gamma]}\{s/x\})$
$= \mathsf{spine}_V(u[e_1\langle\,\vec{w_1}\,\rangle\ldots e_m\langle\,\vec{w_m}\,\rangle \mid c\langle\,\vec{y}\,\rangle\,\overline{[\Gamma]}\{s/x\}])$
$= \mathsf{spine}_V(u[e_1\langle\,\vec{w_1}\,\rangle\ldots e_m\langle\,\vec{w_m}\,\rangle \mid c\langle\,\vec{y}\,\rangle\,\overline{[\Gamma]}]\{s/x\})$ □

We then use SubLemma 79 to prove this next Lemma, which states that when a term is shared we can duplicate the V-spine of it where V is a set of variables free in the term.

Lemma 80. *Given $t \in \Lambda_a^S$, and a set of variables V such that $v \in V \to v \notin (t)_{fc}$*

$$u[x_1,\ldots,x_n \leftarrow t] \rightsquigarrow^*_{R,L,D,C} u\{\mathit{spine}_V(t)^i/x_i\}_{1\le i\le n}\overline{[\Gamma][\Delta]}$$

where the environment $\overline{[\Gamma]}$ is made up of, for each $y \in (t)_{fv} \cap V$, a sharing of the form $[y_1,\ldots,y_m \leftarrow y]$

Proof. We prove this by induction on the sharing weight of t i.e. $\mathcal{W}^1(t)$

Base Case: Variable
$u[x_1,\ldots,x_n \leftarrow z] = u\{y_i/x_i\}_{1\le i\le n}[y_1,\ldots,y_n \leftarrow y]$
$= u\{\mathsf{spine}_V(y)^i/x_i\}_{1\le i\le n}[y_1,\ldots,y_n \leftarrow y]$

Inductive Case: Application
Case: $\mathsf{spine}_V(s\,t) = y$

$u[x_1,\dots,x_n \leftarrow s\,t] = u\{y_i/x_i\}_{i\le n}[y_1,\dots,y_n \leftarrow s\,t]$
$u\{\mathsf{spine}_V(s\,t)^i/x_i\}_{i\le n}[y_1,\dots,y_n \leftarrow s\,t]$

Case: $\mathsf{spine}_V(s\,t) = \mathsf{spine}_V(s)\,y$
$u[x_1,\dots,x_n \leftarrow s\,t] \to_D u\{y_i\,z_i/x_i\}_{i\le n}[y_1,\dots,y_n \leftarrow s][z_1,\dots,z_n \leftarrow t]$
$\overset{\text{I.H.}}{\to^*} u\{y_i\,z_i/x_i\}_{i\le n}\{\mathsf{spine}_V(s)^i/y_i\}_{i\le n}\overline{[\Gamma][\Delta]}[z_1,\dots,z_n \leftarrow t]$
$= u\{\mathsf{spine}_V(s)^i\,z_i/x_i\}_{i\le n}\overline{[\Gamma][\Delta]}[z_1,\dots,z_n \leftarrow t]$
$= u\{\mathsf{spine}_V(s\,t)^i/x_i\}_{i\le n}[\Gamma][\Delta][z_1,\dots,z_n \leftarrow t]$

Case: $\mathsf{spine}_V(s\,t) = y\,\mathsf{spine}_V(t)$
Similar to the case before

Case: $\mathsf{spine}_V(s\,t) = \mathsf{spine}_V(s)\,\mathsf{spine}_V(t)$
$u[x_1,\dots,x_n \leftarrow s\,t] \to_D u\{y_i\,z_i/x_i\}_{i\le n}[y_1,\dots,y_n \leftarrow s][z_1,\dots,z_n \leftarrow t]$
$\overset{\text{I.H.}}{\to^*} u\{y_i\,z_i/x_i\}_{i\le n}\{\mathsf{spine}_V(s)^i/y_i\}_{i\le n}\{\mathsf{spine}_V(t)^i/z_i\}_{i\le n}\overline{[\Gamma_s][\Delta_s][\Gamma_t][\Delta_t]}$
$= u\{\mathsf{spine}_V(s)^i\mathsf{spine}_V(t)^i/x_i\}_{i\le n}[\Gamma_s][\Delta_s][\Gamma_t][\Delta_t] = u\{\mathsf{spine}_V(s\,t)^i/x_i\}_{i\le n}\overline{[\Gamma_s][\Delta_s][\Gamma_t][\Delta_t]}$

Inductive Case: (Phantom-)Abstraction
Case: $\mathsf{spine}_V(c\langle\,\vec{y}\,\rangle.t) = z$
$u[x_1,\dots,x_n \leftarrow c\langle\,\vec{y}\,\rangle.t] = u\{z_i/x_i\}_{i\le n}[z_1,\dots,z_n \leftarrow c\langle\,\vec{y}\,\rangle.t]$
$= u\{\mathsf{spine}_V(c\langle\,\vec{y}\,\rangle.t)^i/x_i\}_{i\le n}[z_1,\dots,z_n \leftarrow c\langle\,\vec{y}\,\rangle.t]$

Case: $\mathsf{spine}_V(c\langle\,\vec{y}\,\rangle.t) = c\langle\,\vec{z}\,\rangle.\mathsf{spine}_V(t)$
$u[x_1,\dots,x_n \leftarrow c\langle\,\vec{y}\,\rangle.t]$
$\rightsquigarrow_D u\{c_i\langle\,w_i\,\rangle.w_i/x_i\}_{i\le n}[c_1\langle\,w_1\,\rangle\dots c_n\langle\,w_n\,\rangle \mid c\langle\,\vec{y}\,\rangle\,[w_1,\dots,w_n \leftarrow t]]$
$\overset{\text{I.H.}}{\to^*} u\{c_i\langle\,w_i\,\rangle.w_i/x_i\}_{i\le n}[c_1\langle\,w_1\,\rangle\dots c_n\langle\,w_n\,\rangle \mid c\langle\,\vec{y}\,\rangle\,\{\mathsf{spine}_V(t)^i/w_i\}_{i\le n}\overline{[\Gamma][\Delta]}]$
$= u\{c_i\langle\,\vec{z}_i\,\rangle.\mathsf{spine}_V(t)^i/x_i\}_{i\le n}[c_1\langle\,\vec{z}_1\,\rangle\dots c_n\langle\,\vec{z}_n\,\rangle \mid c\langle\,\vec{y}\,\rangle\,\overline{[\Gamma][\Delta]}]$
$= u\{\mathsf{spine}_V(c\langle\,\vec{y}\,\rangle.t)^i/x_i\}_{i\le n}[c_1\langle\,\vec{z}_1\,\rangle\dots c_n\langle\,\vec{z}_n\,\rangle \mid c\langle\,\vec{y}\,\rangle\,\overline{[\Gamma][\Delta]}]$
$\rightsquigarrow^*_L u\{\mathsf{spine}_V(c\langle\,\vec{y}\,\rangle.t)^i/x_i\}_{i\le n}\overline{[\Gamma]}[c_1\langle\,\vec{z}_1\,\rangle\dots c_n\langle\,\vec{z}_n\,\rangle \mid c\langle\,\vec{y}\,\rangle\,\overline{[\Delta]}]$
Remark: We can lift $\overline{[\Gamma]}$ since we know $z \in \vec{y} \to z \notin V$

Inductive Case: Sharing
$u[x_1,\dots,x_n \leftarrow s[y_1,\dots,y_m \leftarrow t]] \to_L u[x_1,\dots,x_n \leftarrow s][y_1,\dots,y_m \leftarrow t]$

Case: $\mathsf{spine}_V(s[y_1,\dots,y_m \leftarrow t]) = z$
$u[x_1,\dots,x_n \leftarrow s[y_1,\dots,y_m \leftarrow t]] = u\{z_i/x_i\}_{i\le n}[z_1,\dots,z_n \leftarrow s[y_1,\dots,y_m \leftarrow t]]$
$= u\{\mathsf{spine}_V(s[y_1,\dots,y_m \leftarrow t])^i/x_i\}_{i\le n}[z_1,\dots,z_n \leftarrow s[y_1,\dots,y_m \leftarrow t]]$

Case: $\mathsf{spine}_V(s[y_1,\dots,y_m \leftarrow t]) = \mathsf{spine}_{V\uplus U}(s\sigma)$ where $\sigma = \{t^i/y_i\}_{1\le i\le n}$
$u[x_1,\dots,x_n \leftarrow s][y_1,\dots,y_m \leftarrow t]$
$\overset{\text{I.H.}}{\to^*} u[x_1,\dots,x_n \leftarrow s]\{\mathsf{spine}_V(t)^i/y_i\}_{1\le i\le m}\overline{[\Gamma_t][\Delta_t]}$
$= u[x_1,\dots,x_n \leftarrow s\{\mathsf{spine}_V(t)^i/y_i\}_{1\le i\le m}][\Gamma_t][\Delta_t]$
$\overset{\text{I.H.}}{\to^*} u\{\mathsf{spine}_{V\uplus U}(s\{\mathsf{spine}_V(t)^i/y_i\}_{1\le i\le m})^j/x_j\}_{1\le j\le m}\overline{[\Gamma_s][\Delta_s][\Gamma_t][\Delta_t]}$

$\overset{\text{prop 75}}{=} u\{\mathsf{spine}_{V\uplus U}(s\{\mathsf{spine}_{V^i}(t^i)/y_i\}_{1\le i\le m})^j/x_j\}_{1\le j\le m}\overline{[\Gamma_s][\Delta_s][\Gamma_t][\Delta_t]}$
$= u\{\mathsf{spine}_{V\uplus U}(s\{t^i/y_i\}_{1\le i\le m})^j/x_j\}_{1\le j\le m}\overline{[\Gamma_s][\Delta_s][\Gamma_t][\Delta_t]}$
Since $\mathsf{spine}_V(-)$ is idempotent and V^i is contained in U
$= u\{\mathsf{spine}_V(s[y_1,\ldots,y_m \leftarrow t])^j/x_j\}_{1\le j\le m}\overline{[\Gamma_s][\Delta_s][\Gamma_t][\Delta_t]}$
Lastly, perform any $\rightarrow_L$ rules required in the sharings $\overline{[\Gamma_s][\Gamma_t]}$ to obtain $\overline{[\Gamma]}$

Inductive Case: Distributor
$u[x_1,\ldots,x_n \leftarrow s[e_1\langle\,\vec{w}_1\,\rangle\ldots e_m\langle\,\vec{w_m}\,\rangle\,|\,c\langle\,\vec{y}\,\rangle\,\overline{[\Sigma]}]]$

Let W be such that
$= \mathsf{spine}_{V\uplus W}(s\sigma) = \mathsf{spine}_V(s\overline{[\Sigma]}) = \mathsf{spine}_V(s[e_1\langle\,\vec{w}_1\,\rangle\ldots e_n\langle\,\vec{w}_n\,\rangle\,|\,c\langle\,\vec{y}\,\rangle\,\overline{[\Sigma]}])$

Let $[a^1_1,\ldots,a^1_{k_1} \leftarrow t_1]\ldots[a^m_1,\ldots,a^m_{k_m} \leftarrow t_m]$ be all the sharings in $\overline{[\Sigma]}$, including those in nested distributors, where we maintain the order of the sharings in the term calculus.
We apply the induction hypothesis on the m^{th} sharing (the outermost).
$[a^1_1,\ldots,a^1_{k_1} \leftarrow t_1]\ldots[a^{m-1}_1,\ldots,a^{m-1}_{k_{m-1}} \leftarrow t_{m-1}][a^m_1,\ldots,a^m_{k_m} \leftarrow t_m]$
$\overset{\text{I.H.}}{\rightarrow^*} [a^1_1,\ldots,a^1_{k_1} \leftarrow t_1]\ldots[a^{m-1}_1,\ldots,a^{m-1}_{k_{m-1}} \leftarrow t_{m-1}]\{\mathsf{spine}_V(t_m)^i/a^m_i\}_{1\le i\le k_m}\overline{[\Gamma_m][\Delta_m]}$
$= \sigma_m[a^1_1,\ldots,a^1_{k_1} \leftarrow t_1\sigma_m]\ldots[a^{m-1}_1,\ldots,a^{m-1}_{k_{m-1}} \leftarrow t_{m-1}\sigma_m]\overline{[\Gamma_m][\Delta_m]}$
where $\sigma_m = \{\mathsf{spine}_{V^i}({t_m}^i)/a^m_i\}_{1\le i\le k_m}$ by Prop 75
Now we can apply the induction hypothesis on the $m-1^{\text{th}}$ sharing

$\overset{\text{I.H.}}{\rightarrow^*} \sigma_m[a^1_1,\ldots,a^1_{k_1} \leftarrow t_1\sigma_m]\ldots[a^{m-1}_1,\ldots,a^{m-2}_{k_{m-2}} \leftarrow t_{m-2}\sigma_m]$
$\{\mathsf{spine}_{V\uplus V^1\uplus\cdots\uplus V^{k_m}}(t_{m-1}\sigma_m)^i/a^{m-1}_i\}_{1\le i\le k_{m-1}}\overline{[\Gamma_{m-1}][\Delta_{m-1}][\Gamma_m][\Delta_m]}$

$\overset{\text{lem 79}}{=} [a^1_1,\ldots,a^1_{k_1} \leftarrow t_1\sigma_m]\ldots[a^{m-1}_1,\ldots,a^{m-2}_{k_{m-2}} \leftarrow t_{m-2}\sigma_m]$
$\{\mathsf{spine}_{V\uplus V^1\uplus\cdots\uplus V^{k_m}}(t_{m-1}\sigma'_m)^i/a^{m-1}_i\}_{1\le i\le k_{m-1}}\overline{[\Gamma_{m-1}][\Delta_{m-1}][\Gamma_m][\Delta_m]}$
where $\sigma'_m = \{{t_m}^i/a^m_i\}_{1\le i\le k_m}$

Continuing until we apply the induction hypothesis to all sharings, we reach
$\sigma_m\sigma_{(m-1,m)}\ldots\sigma_{(1,\ldots,m-1,m)}\overline{[\Gamma_1][\Delta_1]}\ldots\overline{[\Gamma_{m-1}][\Delta_{m-1}][\Gamma_m][\Delta_m]}$
Where for example $\sigma_{(m-1,m)} = \{\mathsf{spine}_{V\uplus V^1\uplus\cdots\uplus V^{k_m}}(t_{m-1}\sigma'_m)^i/a^{m-1}_i\}_{1\le i\le k_{m-1}}$
Note: By SubLemma 79, each substitution can be represented with one application of $\mathsf{spine}_{V\uplus W}(-)$

Let $\overline{[\Sigma']}$ be $\overline{[\Sigma]}$ where every sharing $[a^i_1,\ldots,a^i_{k_i} \leftarrow t_i]$ is replaced with $\overline{[\Gamma_i][\Delta_i]}$
Then $u[x_1,\ldots,x_n \leftarrow s[e_1\langle\,\vec{w}_1\,\rangle\ldots e_m\langle\,\vec{w_m}\,\rangle\,|\,c\langle\,\vec{y}\,\rangle\,\sigma_m\sigma_{(m-1,m)}\ldots\sigma_{(1,\ldots,m-1,m)}\overline{[\Sigma']}]]$
$= u[x_1,\ldots,x_n \leftarrow s\sigma_m\sigma_{(m-1,m)}\ldots\sigma_{(1,\ldots,m-1,m)}[e_1\langle\,\vec{w'_1}\,\rangle\ldots e_m\langle\,\vec{w'_m}\,\rangle\,|\,c\langle\,\vec{y}\,\rangle\,\overline{[\Sigma']}]]$
$\rightarrow_L u[x_1,\ldots,x_n \leftarrow s\sigma_m\sigma_{(m-1,m)}\ldots\sigma_{(1,\ldots,m-1,m)}][e_1\langle\,\vec{w'_1}\,\rangle\ldots e_m\langle\,\vec{w'_m}\,\rangle\,|\,c\langle\,\vec{y}\,\rangle\,\overline{[\Sigma']}]$
$\overset{\text{I.H.}}{\rightarrow^*} u\{\mathsf{spine}_{V\uplus W}(s\sigma_m\sigma_{(m-1,m)}\ldots\sigma_{(1,\ldots,m-1,m)})^i/x_i\}_{1\le i\le n}\overline{[\Gamma_s][\Delta_s]}[e_1\langle\,\vec{w'_1}\,\rangle\ldots e_m\langle\,\vec{w'_m}\,\rangle\,|\,c\langle\,\vec{y}\,\rangle\,\overline{[\Sigma']}]$

$\mathsf{spine}_{V\uplus W}(s\sigma_m\sigma_{(m-1,m)}\ldots\sigma_{(1,\ldots,m-1,m)})$
$= \mathsf{spine}_V(s\overline{[\Sigma]}) = \mathsf{spine}_V(s[e_1\langle\,\vec{w}_1\,\rangle\ldots e_n\langle\,\vec{w}_n\,\rangle\,|\,c\langle\,\vec{y}\,\rangle\,[\Sigma]])$

Lastly, we lift $\rightarrow_L$ each $\overline{[\Gamma_i]}$ out of the distributors, and we know we can always do this by conditions of the Lemma i.e. $\nexists_{v \in V}\ (v \in \vec{y})$
Let $\overline{[\Sigma'']}$ be $\overline{[\Sigma']}$ with all sharings $\overline{[\Gamma_i]}$ removed
$\rightarrow_L^* u\{\mathsf{spine}_V(s[e_1\langle\, \vec{w}_1 \,\rangle \ldots e_n\langle\, \vec{w}_n \,\rangle \,|\, c\langle\, \vec{y} \,\rangle\, \overline{[\Sigma]}])^i / x_i\}_{1 \leq i \leq n}$
$\overline{[\Gamma][\Delta_s]}[e_1\langle\, \vec{w'_1} \,\rangle \ldots e_m\langle\, \vec{w'_m} \,\rangle \,|\, c\langle\, \vec{y} \,\rangle\, \overline{[\Sigma'']}]$
□

Theorem 81. *The spinal atomic λ-calculus Λ_a^S implements spine duplication: a term $u[x_1, \ldots, x_n \leftarrow c\langle\, c \,\rangle.t]$ reduced to a term of the form $u\{(c\langle\, c \,\rangle.\mathit{spine}_{\{c\}}(t))^i / x_i\}_{1 \leq i \leq n}[\Delta]$*

Proof. By Lemma 80 □

Chapter 8

Conclusion

思前想后
sī qián xiǎng hòu
To accurately ponder over something

8.1 What we have

This dissertation describes a typed λ-calculus with explicit sharing that implements spine duplication through atomic reduction steps. There is a natural graphical interpretation. This calculus is the result of stuyding the Curry-Howard interpretation of open deduction provided in [GHP13] extended with a new inference rule that corresponds to scope in the term calculus.

The original calculus exposed a connection between proof reduction in deep inference and the sharing mechanisms of optimal reduction graphs. Our result further expands this connection. The sharing mechanisms of optimal reduction graphs include scope manipulation. In our calculus, we manipulate the scopes of an abstraction during duplication (in the distributor), which allows us to duplicate only the spine of the term. It is in Chapter 7 where we prove that this calculus can duplicate the spine of the body of an abstraction.

We discuss in Chapter 2 the correspondence between the switch rule and an end-of-scope operator, such as adbmal ($\curlywedge$) and director strings. In the case of adbmal, the scope of an abstraction becomes explicitly indicated in the term. This opens up a distinction between *balanced* and *unbalanced* scopes: whether scopes must be properly nested, or not; for example, in $\lambda x.\lambda y.N$, a subterm $\curlywedge y.\curlywedge x.M$ is balanced, but $\curlywedge x.\curlywedge y.M$ is not. With balanced scope, one can indicate the skeleton of an abstraction; with unbalanced scope (which Hendriks and Van Oostrom dismiss) one can indicate the spine. A closely related approach is *director strings*. The idea is to use nameless abstractions identified by their nesting (as with De Bruijn indices), and make the paths to bound variables explicit by annotating each constructor with a string of *directors*. The primary aim of these approaches is to eliminate α-conversion and to streamline substitution. Consequently, while they can *identify* the spine, they do not readily isolate it for duplication, as would be required for spinal full laziness. Furthermore, atomic duplication allows for the natural duplication of the spine, whereas extracting the spine of a term making use of director strings will involve

a lot of overhead and maintenance of annotations, something which is limited in our result (only when lifting a closure out of a distributor is book-keeping required).

The main result of this work is preservation of strong normalization, which means that if a term $M \in \Lambda$ is strongly normalising then its interpretation $(\!|\, M \,|\!)$ in the spine calculus is strongly normalising. We prove this from the results of other chapters. The main result of Chapter 5 tells us that sharing reductions are strongly normalising and confluent, which means that a term with an infinite reduction sequence must have infinitely many β steps. We prove strong normalisation of sharing reductions by constructing a measure that maps constructors of terms in the atomic λ-calculus onto constructors of the equivalent term in the weakening calculus (introduced in Chapter 4), and proving this measure strictly decreases during reduction.

We know that infinite reduction can happen in the atomic λ-calculus, e.g. the term $t = u[\leftarrow (x\langle\, x \,\rangle.x\, x)\, z\langle\, z \,\rangle.z\, z]$ has an infinite reduction, specifically inside the weakened terms, which is discarded in their readback interpretation $[\![\, t \,]\!]$. Discarding this reduction sequence creates a potential problem, as by contraposition PSN states that if a term $(\!|\, M \,|\!) \in \Lambda_a^S$ has an infinite reduction sequence sequence then M has an infinite reduction sequence, and compilation may introduce weakenings e.g. $(\!|\, \lambda y.M \,|\!)$ where $y \notin (M)_{fv}$. Therefore throwing away the weakenings will thwart the direct construction of an infinite reduction sequence in the λ-calculus from an infinite atomic reduction.

Using the same solution to this problem as used in [GHP13], we utilise an intermediate calculus, the weakening calculus. It has already been shown that the weakening calculus, a calculus with explicit weakenings, satisfies PSN with respect to the λ-calculus. Therefore we prove PSN for the atomic λ-calculus with respect to the weakening calculus, which consequently proves PSN.

8.2 Steps forward

There are many directions in which we can take our work. Firstly, one could develop an implementation of the λ-calculus based on the (spinal) atomic λ-calculus. This implementation would use environments to implement sharing, and can perform full laziness. One would need to choose a β-reduction strategy. Lazy evaluation (call-by-need) would be a sensible first choice, but complete laziness is also feasible. It would be interesting to see an implementation making use of the atomic reduction rules that implements complete laziness, as it would allow us to study these more complicated reduction strategies in a form other than graph rewrite rules as well as further demonstrate the differences between the duplication strategies and evaluation strategies.

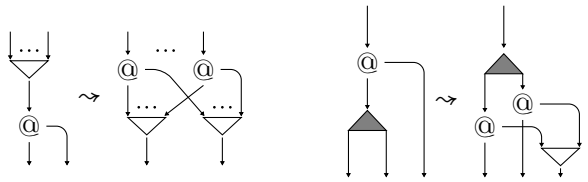
However, some important properties to achieve such an implementation are missing. Reduction is non-local: several reduction steps involve the condition that a given variable is not free in a given subterm, which requires inspection of the entire subterm. As an example, observe rule (l_3). Due to the nature of reduction, this cannot easily be addressed by a simple one-pass annotation of terms with free variables, since terms are manipulated at the atomic level and free variables are in constant flux. Variable location is even required in a graphical implementation. Duplicating the spine of an abstraction $x\langle\, x \,\rangle.t$ means knowing precisely the location of the free variable x in t, and duplicating accordingly. Therefore, one would have to decide if a sharing node in a graph (or a sharing in the term calculus)

would need to proceed duplicating or to stop (where in the term calculus, we would lift the closure out of the distributor with either l_5 or l_6).

There are many abstract machines and implementations of the λ-calculus, and sometimes it is not feasible to make comparisons on their behaviors (perhaps due to different methods of implementation such as graphical or using environments). Due to having a close syntax to the λ-calculus, and working with environments while having a strong graphical intuition, the atomic λ-calculus is a good candidate to unionize these different approaches to sharing.

Additionally our calculus has a close relationship to calculus with explicit substitution [ACCL91]. Typically these calculi are used to obtain more control over the substitution process to allow for a deeper understanding of the execution models of higher-order languages. A calculus was explicit substitutions, and explicit constructs of weakening and binary sharing was proven in [KL07] to preserve strongly normalising terms. One difference is that this calculus uses two constructs to differentiate weakenings and sharings, whereas in our calculus these are distinguishable instances of the same construct. Furthermore only variables may be shared, and a term t is semantically shared by sharing a variable x and using an explicit substitution that would replace x with term t. The biggest difference however is that duplication of terms in our calculus proceeds atomically, allowing for more control over not only the substitution process but also the duplication process. Explicit substitutions, like our calculus, also have a strong graphical intuition which has been formalised in [DCKP00].

Graphical intuitions have been given for terms (Figure 3.3 and 3.4) and reduction rules (Figure 3.6, 3.7, 3.8, and 3.9). However, these intuitions do not provide the full system, i.e. they have a correspondence to the structural rules of our system (e.g. application) but not the logical rules (e.g. switch). A graphical formalism may help provide a deeper insight into the calculus, since we would expect the derivation of a term in open deduction to correspond to the graph, therefore the typing system is more easily observed. Furthermore, studying the relationships between graphical systems (such as Lamping's graphs) and our system becomes easier, as for example, the difference in how we duplicate an application as shown in Figure 8.1, where Figure 8.1a shows how applications are duplicated in our calculus compared to how they are duplicated in all reduction strategies for optimal reduction in Figure 8.1b.



(a) Atomic graphs (b) Sharing graphs

Figure 8.1: Duplication of application

Another direction would be to translate the result of Blanc, Lévy, and Maranget [BLM07] into our calculus. There they show optimal reduction for a lambda calculus with a weaker version of β-reduction. If we show their result in our formalism, we may develop a founda-

tions to a logical framework for optimal reduction.

Lastly, and a long-term goal, would be to develop a complete logical framework for optimal reduction. The algorithms for optimal reduction us graph rewriting techniques, and nodes that specify enclosures and scopes. Furthermore, the implementations do not share terms like in either of the atomic λ-calculi, but they share contexts i.e. terms with holes that can be 'filled' with distinct terms. Thus it is clear that they use a notion of sharing more powerful than full laziness.

It is worth mentioning a step in this direction has been made in [GM13] where Gimenez and Moser compare the rewrite rules for a multiplicative exponential linear logic deep inference system to graph rewrite rules (specifically interaction nets [Laf90]) used for optimal reduction, however not every graphical reduction has a corresponding logical equivalent step. It would be interesting to discover the Curry-Howard correspondence between the nodes used for optimal sharing (brackets and croissants) and their logical interpretation in deep inference, and help formalise the requirements for implementing optimal reduction.

Bibliography

[ACCL91] Martin Abadi, Luca Cardelli, Pierre-Loius Curien, and Jean-Jacques Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.

[AF97] Zena M. Ariola and Matthias Felleisen. The call-by-need lambda calculus. *Journal of Functional Programming*, 7(3):265–301, 1997.

[AK12a] Beniamino Accattoli and Delia Kesner. The permutative λ-calculus. In Nikolaj Bjørner and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 23–36, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[AK12b] Beniamino Accattoli and Delia Kesner. Preservation of Strong Normalisation modulo permutations for the structural lambda-calculus. *Logical Methods in Computer Science*, 8(1):44, March 2012.

[AT17] Andrea Aler Tubella. *A Study of Normalisation Through Subatomic Logic*. PhD thesis, University of Bath, 2017.

[Bal12] Thibaut Balabonski. A unified approach to fully lazy sharing. *SIGPLAN Not.*, 47(1):469–480, January 2012.

[Bar84] Henk P Barendregt. The lambda calculus: Its syntax and semantics, revised ed., vol. 103 of studies in logic and the foundations of mathematics, 1984.

[BF82] Klaus J. Berkling and Elfriede Fehr. A consistent extension of the lambda-calculus as a base for functional programming languages. *Information and Control*, 55(1):89 – 101, 1982.

[BKKS87] H.P. Barendregt, J.R. Kennaway, J.W. Klop, and M.R. Sleep. Needed reduction and spine strategies for the lambda calculus. *Information and Computation*, 75(3):191 – 231, 1987.

[BL05] Kai Brünnler and Stéphane Lengrand. On two forms of bureaucracy in derivations. In Paola Bruscoli, François Lamarche, and Charles Stewart, editors, *Structures and Deduction*, pages 69–80. Technische Universität Dresden, 2005. ICALP Workshop. ISSN 1430-211X.

[BLM07] Tomasz Blanc, Jean-Jacques Lévy, and Luc Maranget. Sharing in the weak lambda-calculus revisited. In Erik Barendsen, Herman Geuvers, Venanzio

Capretta, and Milad Niqui, editors, *Reflections on Type Theory, Lambda Calculus, and the Mind, Essays Dedicated to Henk Barendregt on the Occasion of his 60th Birthday*, pages 41–50. Nijmegen Radboud Universiteit Nijmegen, 2007.

[Brü06] Kai Brünnler. Deep sequent systems for modal logic. In Guido Governatori, Ian Hodkinson, and Yde Venema, editors, *Advances in Modal Logic (AiML)*, volume 6, pages 107–119. College Publications, 2006.

[Brü10] Kai Brünnler. Nested sequents. Habilitation Thesis, Universität Bern, 2010.

[BT01] Kai Brünnler and Alwen Fernanto Tiu. A local system for classical logic. In R. Nieuwenhuis and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 2250 of *Lecture Notes in Computer Science*, pages 347–361. Springer-Verlag, 2001.

[CDC78] M. Coppo and M. Dezani-Ciancaglini. A new type assignment for λ-terms. *Archiv für mathematische Logik und Grundlagenforschung*, 19(1):139–156, 1978.

[CDC80] M. Coppo and M. Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre Dame J. Formal Logic*, 21(4):685–693, 1980.

[CH98] Naim Cagman and J.Roger Hindley. Combinatory weak reduction in lambda calculus. *Theoretical Computer Science*, 198(1):239 – 247, 1998.

[Chu41] Alonzo Church. *The Calculi of Lambda-Conversion*. Princeton University Press, 1941.

[CR36] Alonzo Church and J. Barkley Rosser. Some properties of conversion. *Transactions of the American Mathematical Society*, 39(3):472–482, 1936.

[CS97] J.R.B. Cockett and R.A.G. Seely. Weakly distributive categories. *Journal of Pure and Applied Algebra*, 114(2):133 – 173, 1997.

[Cur30] H. B. Curry. Grundlagen der kombinatorischen logik. *American Journal of Mathematics*, 52(3):509–536, 1930.

[Das14a] Anupam Das. *The Complexity of Propositional Proofs in Deep Inference*. PhD thesis, University of Bath, 2014.

[Das14b] Anupam Das. On the pigeonhole and related principles in deep inference and monotone systems. In Thomas Henzinger and Dale Miller, editors, *Joint Meeting of the 23rd EACSL Annual Conference on Computer Science Logic (CSL) and the 29th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 36:1–10. ACM, 2014.

[dB72] Nicolaas Govert de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. In *Indagationes Mathematicae (Proceedings)*, volume 75, pages 381–392. Elsevier, 1972.

[DCKP00] Roberto Di Cosmo, Delia Kesner, and Emmanuel Polonovski. Proof nets and explicit substitutions. In Jerzy Tiuryn, editor, *Foundations of Software Science and Computation Structures*, pages 63–81, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[DG01] René David and Bruno Guillaume. A λ-calculus with explicit weakening and explicit substitution. *Mathematical Structures in Computer Science*, 11(1):169–206, 2001.

[DP04] Kosta Došen and Zoran Petrić. *Proof-theoretical coherence*. King's College Publications, 35 Ballards Lane, London N3 1XW, 2004.

[FM99] Maribel Fernández and Ian Mackie. Closed reductions in the λ-calculus. In Jörg Flum and Mario Rodriguez-Artalejo, editors, *Computer Science Logic*, pages 220–234, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[FMS05a] Maribel Fernández, Ian Mackie, and François-Régis Sinot. Closed reduction: explicit substitutions without α-conversion. *Mathematical Structures in Computer Science*, 15(2):343–381, 2005.

[FMS05b] Maribel Fernández, Ian Mackie, and François-Régis Sinot. Lambda-calculus with director strings. *Applicable Algebra in Engineering, Communication and Computing*, 15(6):393–437, Apr 2005.

[FPT99] M. Fiore, G. Plotkin, and D. Turi. Abstract syntax and variable binding. In *Proceedings. 14th Symposium on Logic in Computer Science (Cat. No. PR00158)*, pages 193–202, July 1999.

[GAL92] Georges Gonthier, Martín Abadi, and Jean-Jacques Lévy. The geometry of optimal lambda reduction. In *Proceedings of the 19th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '92, pages 15–26, New York, NY, USA, 1992. ACM.

[GG08] Alessio Guglielmi and Tom Gundersen. Normalisation control in deep inference via atomic flows. *Logical Methods in Computer Science*, 4(1):9:1–36, 2008.

[GGP10] Alessio Guglielmi, Tom Gundersen, and Michel Parigot. A proof calculus which reduces syntactic bureaucracy. In Christopher Lynch, editor, *21st International Conference on Rewriting Techniques and Applications (RTA)*, volume 6 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 135–150. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2010.

[GHP13] Tom Gundersen, Willem Heijltjes, and Michel Parigot. Atomic lambda calculus: A typed lambda-calculus with explicit sharing. In Orna Kupferman, editor, *28th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 311–320. IEEE, 2013.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1 – 101, 1987.

[GM13] Stéphane Gimenez and Georg Moser. The Structure of Interaction. In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013 (CSL 2013)*, volume 23 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 316–331, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[GS01] Alessio Guglielmi and Lutz Straßburger. Non-commutativity and mell in the calculus of structures. In Laurent Fribourg, editor, *Computer Science Logic*, pages 54–68, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[GS14] Nicolas Guenot and Lutz Straßburger. Symmetric normalisation for intuitionistic logic. In Thomas Henzinger and Dale Miller, editors, *Joint Meeting of the 23rd EACSL Annual Conference on Computer Science Logic (CSL) and the 29th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 45:1–10. ACM, 2014.

[GS17] Stefano Guerrini and Marco Solieri. Is the Optimal Implementation Inefficient? Elementarily Not. In *2nd International Conference on Formal Structures for Computation and Deduction (FSCD 2017)*, volume 84, pages 17:1 – 17:16, Oxford, United Kingdom, September 2017.

[Gug07] Alessio Guglielmi. A system of interaction and structure. *ACM Transactions on Computational Logic*, 8(1):1:1–64, 2007.

[He18] Fanny He. *The Atomic Lambda-Mu Calculus*. PhD thesis, University of Bath, 2018.

[HG91] Carsten Kehler Holst and Darsten Krogh Gomard. Partial evaluation is fuller laziness. *SIGPLAN Not.*, 26(9):223–233, May 1991.

[HO80] Gérard Huet and Derek C. Oppen. Equations and rewrite rules: A survey. In RONALD V. BOOK, editor, *Formal Language Theory*, pages 349 – 405. Academic Press, 1980.

[How80] William A Howard. The formulae-as-types notion of construction. *To H. B. Curry : Essays on combinatory logic, lambda-calculus, and formalism*, 44:479–490, 1980.

[Hug04] Dominic J.D. Hughes. Deep inference proof theory equals categorical proof theory minus coherence, 2004.

[HvO03] Dimitri Hendriks and Vincent van Oostrom. adbmal. In Franz Baader, editor, *Automated Deduction - CADE-19, 19th International Conference on Automated Deduction Miami Beach, FL, USA, July 28 - August 2, 2003, Proceedings*, volume 2741 of *Lecture Notes in Computer Science*, pages 136–150, 2003.

[JL82] Jean-Pierre Jouannaud and Pierre Lescanne. On multiset orderings. *Information Processing Letters*, 15(2):57 – 63, 1982.

[Joh85] Thomas Johnsson. Lambda lifting: Transforming programs to recursive equations. In *Proc. Of a Conference on Functional Programming Languages and Computer Architecture*, pages 190–203, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[KC08] Delia Kesner and Shane Ó Conchúir. Milner's lambda calculus with partial substitutions. *Available on http://www. pps. jussieu. fr/ kesner/papers Last visited on 07/01/2019*, 2008.

[Kes08] Delia Kesner. Perpetuality for full and safe composition (in a constructive setting). In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming*, pages 311–322, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[Kes09] Delia Kesner. A Theory of Explicit Substitutions with Safe and Full Composition. *Logical Methods in Computer Science*, 5(3:1):Pages 1–29, 2009.

[KL05] Delia Kesner and Stéphane Lengrand. Extending the explicit substitution paradigm. In Jürgen Giesl, editor, *Term Rewriting and Applications*, pages 407–422, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[KL07] Delia Kesner and Stéphane Lengrand. Resource operators for λ-calculus. *Information and Computation*, 205(4):419 – 473, 2007. Special Issue: 16th International Conference on Rewriting Techniques and Applications.

[Klo80] Jan Willem Klop. *Combinatory Reduction Systems*. PhD thesis, Utrecht University, 1980.

[KS88] Richard Kennaway and Ronan Sleep. Director strings as combinators. *ACM Trans. Program. Lang. Syst.*, 10(4):602–626, October 1988.

[KvOvR93] Jan Willem Klop, Vincent van Oostrom, and Femke van Raamsdonk. Combinatory reduction systems: introduction and survey. *Theoretical Computer Science*, 121(1):279 – 308, 1993.

[Laf90] Yves Lafont. Interaction nets. In *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '90, pages 95–108, New York, NY, USA, 1990. ACM.

[Lam90] John Lamping. An algorithm for optimal lambda calculus reduction. In *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '90, pages 16–30, New York, NY, USA, 1990. ACM.

[Lau93] John Launchbury. A natural semantics for lazy evaluation. In *Proceedings of the 20th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '93, pages 144–154, New York, NY, USA, 1993. ACM.

[Lév80] Jean-Jacques Lévy. Optimal reductions in the lambda calculus. *To HB Curry: Essays on Combinatory Logic, Lambda Coalculus and Formalism*, pages 159–191, 1980.

[Mac98] Ian Mackie. Yale: Yet another lambda evaluator based on interaction nets. *SIGPLAN Not.*, 34(1):117–128, September 1998.

[Mel95] Paul-André Mellies. Typed λ-calculi with explicit substitutions may not terminate. In Mariangiola Dezani-Ciancaglini and Gordon Plotkin, editors, *Typed Lambda Calculi and Applications*, pages 328–334, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[PH18] Joseph Paulus and Willem Heijltjes. Deep-inference intersection types. *Presented at Twenty Years of Deep Inference Workshop 2018, Oxford. Available on http://www.cs.bath.ac.uk/wbh22/pdf/2018-heijltjes-paulus.pdf Last visited on 07/01/2019*, 2018.

[PJ87] Simon Peyton Jones. *The Implementation of Functional Programming Languages*. Prentice Hall, January 1987.

[Roj15] Raúl Rojas. A tutorial introduction to the lambda calculus. *arXiv preprint arXiv:1503.09060*, 2015.

[Ses97] Peter Sestoft. Deriving a lazy abstract machine. *Journal of Functional Programming*, 7(3):231–264, 1997.

[SFM03] François-Régis Sinot, Maribel Fernández, and Ian Mackie. Efficient reductions with director strings. In Robert Nieuwenhuis, editor, *Rewriting Techniques and Applications*, pages 46–60, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[Sin06] François-Régis Sinot. Call-by-need in token-passing nets. *Mathematical Structures in Computer Science*, 16(4):639–666, 2006.

[Sin08] François-Régis Sinot. Complete laziness: a natural semantics. *Electronic Notes in Theoretical Computer Science*, 204:129 – 145, 2008. Proceedings of the 7th International Workshop on Reduction Strategies in Rewriting and Programming (WRS 2007).

[SS05] Charles Stewart and Phiniki Stouppa. A systematic proof theory for several modal logics. In Renate Schmidt, Ian Pratt-Hartmann, Mark Reynolds, and Heinrich Wansing, editors, *Advances in Modal Logic (AiML)*, volume 5, pages 309–333. King's College Publications, 2005.

[Sto07] Phiniki Stouppa. A deep inference system for the modal logic S5. *Studia Logica*, 85(2):199–214, 2007.

[Str07] Lutz Straßburger. A characterization of medial as rewriting rule. In Franz Baader, editor, *Term Rewriting and Applications*, pages 344–358, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[SW04] Olin Shivers and Mitchell Wand. Bottom-up beta-substitution: Uplinks and lambda-dags. *BRICS Report Series*, 11(38), Dec. 2004.

[Tiu06] Alwen Tiu. A local system for intuitionistic logic. In Miki Hermann and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 242–256, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[vOvdLZ04] Vincent van Oostrom, Kees-Jan van de Looij, and Marijn Zwitserlood. Lambdascope: another optimal implementation of the lambda-calculus. In *Workshop on Algebra and Logic on Programming Systems (ALPS)*, 2004.

[Wad71] Christopher P. Wadsworth. *Semantics and Pragmatics of the Lambda-Calculus.* PhD thesis, University of Oxford, 1971.

[WR28] Alfred North Whitehead and Bertrand Russell. Principia mathematica. *Journal of Philosophy*, 25(16):438–445, 1928.